Port, component, signal の説明　Ver.3.0.0.0

･ife.vhd (Instruction Fetch)

･･Port

Port(

clk　　: in std_logic;

siki から送られてくるクロック

clk (siki.vhd) => clk

state　　: in std_logic_vector(1 downto 0);

siki で管理されている state

state が”00”で BRAM に PC を読み込みアドレスを入れ、”10”で読み込まれた次の命令を next_instr に格納し”11”で ID に渡している。

state (siki.vhd) => state

exec_mode　: in std_logic; --activate pipeline mode when '1'

'1'になっているとパイプライン処理に変更できる

(現在は未実装)

'0'のときは連続処理(微妙にパイプライン化されているが)を行う

(現在はこちらを使用中)

pipeline_mode (siki.vhd) => exec_mode

write_mem　: in std_logic; --write BRAM(write: '1', read: '0')

WEA(Write EnAble)とつながっており、siki から命令データを BRAM に格納するかのシグナルを受け取る。

これが立っているときはプロセス文が命令データ読み込みサイクルに入り、下りているときは命令実行モードになっている。

'1'で書き込み、'0'で読み込み

load_instr (siki.vhd) => write_mem

write_BRAM　: in std_logic_vector(15 downto 0); --write to this BRAM address

命令データを受け取るとき siki から命令を書き込むアドレスをここに格納し、BRAM の ADDRA(Address A)に伝える

write_BRAM (siki.vhd) => write_BRAM

next_PC　　: in std_logic_vector(15 downto 0);

これから読み込む命令の PC

最初は siki から初期 PC が送られ、その後は ALU から次の PC が送られる。

next_PC (siki.vhd) => next_PC

instruction : in std_logic_vector(31 downto 0); --data to BRAM

siki から送られる BRAM に書き込まれる命令データ

full_recv (siki.vhd) => instruction

prev_PC : out std_logic_vector(15 downto 0); --previous PC
id に現在処理している命令の PC を渡している。
prev_PC => write_PC (siki.vhd) => write_PC (siki.vhd)

recv_data : out std_logic_vector(31 downto 0)); --data from BRAM
BRAM から読み込んだ命令
recv_data => instr (siki.vhd) => instr (id.vhd)

・・**Component**

```
component BRAM
 port(
  ADDRA : in std_logic_vector(15 downto 0); --address of BRAM
```

ADDRess A のこと
BRAM にアクセスする際のアドレス
基本的に PC がアドレスとして用いられている。
ADDRA => addra_buf

```
  DINA  : in std_logic_vector(31 downto 0); --data in
```

Data IN A のこと
BRAM に書き込むデータ
読み込まれた命令データか書き込まれる
DINA => dina_buf

```
  WEA   : in std_logic_vector(0 downto 0); --write enable(write: '1', read: '0')
```

Write Enable A のこと
'1'のとき BRAM に DINA の内容が書き込まれ、'0'のとき DOUTA にデータが送られる。
WEA => wea_buf

```
  CLKA  : in std_logic;
```

CLocK A のこと
BRAM に渡している clk
CLKA => clk

```
  DOUTA : out std_logic_vector(31 downto 0)); --data out
```

Data OUT A のこと
命令処理中に次に処理する命令が BRAM から送られてくる。
DOUT => douta_buf

・・**Signal**

```
signal prev_PC_buf   : std_logic_vector(15 downto 0) := x"0000";
signal recv_data_buf : std_logic_vector(31 downto 0) := x"00000000";
```

バッファ群
state が”11”になったとき ID にデータを渡している。

```
signal addra_buf : std_logic_vector(15 downto 0) := x"0000";
signal dina_buf  : std_logic_vector(31 downto 0) := x"00000000";
signal wea_buf   : std_logic_vector(0 downto 0) := "0";
signal douta_buf : std_logic_vector(31 downto 0) := x"00000000";
```

BRAM とのやりとりで用いられるバッファ

```
signal PC : std_logic_vector(15 downto 0) := x"0000"; --previous PC
```

next_PC と直結
現在処理している PC が格納されている。
(不要な可能性あり)

```
signal next_instr : std_logic_vector(31 downto 0) := x"00000000"; --instruction
```

命令処理中に douta_buf から次の命令を読み取り保持する。

```
signal write_data : std_logic_vector(31 downto 0) := x"00000000"; --write to BRAM
```

instruction と直結
siki が受け取った命令データを BRAM に渡す。
(不要な可能性あり、というか不要、というか使われてないじゃん!)

```
signal delay_state   : std_logic_vector(1 downto 0) := "00"; --sequential modes state (next instruction starts when previous instruction is at memory write state)
```

命令を連続処理モードで処理している時に次の命令を実行するタイミングまでこれで実行を停止させる。
“00”,”01”,”10”,”00”,…と遷移し、”00”となっているときに次の命令を ID に送る。

・id.vhd (Instruction Decode & Write Register)

・・Port

```
Port(
  clk               : in std_logic;
```

siki から送られてくるクロック
clk (siki.vhd) => clk

```
  state             : in std_logic_vector(1 downto 0);
```

siki で管理されている state
“11”になったとき ALU にデータを渡し、更にレジスタの中身を更新する
state (siki.vhd) => state

```
  write_PC          : in std_logic_vector(15 downto 0); --(present)PC
```

現在処理している命令の PC
ALU にそのまま渡したり data_1 に渡したりする
prev_PC(ife.vhd) => write_PC (siki.vhd) => write_PC

```
  instr             : in std_logic_vector(31 downto 0);
```

ife から送られてきた次の命令
この instr を各パーツに分解し、データを得る
recv_data (ife.vhd) => instr (siki.vhd) => instr

```
  mem_to_R_sig_return : in std_logic; --signal from memory whether to write
```

外部から送られてくるレジスタに書き込むかのシグナル
基本的に mem から送られてくるが初期値設定やシステムコールで siki が介入することがある。
mem_to_R_sig_return (siki.vhd) => mem_to_R_sig_return

```
  mem_to_R_pointer    : in std_logic_vector(4 downto 0); --write data to this register
```

外部から送られてくるどのレジスタに書き込むかを指示するシグナル
基本的に mem から送られてくるが初期値設定やシステムコールで siki が介入することがある。
mem_to_R_pointer (siki.vhd) => mem_to_R_pointer

```
  mem_to_R          : in std_logic_vector(31 downto 0); --data from memory
```

外部から送られてくるレジスタに書き込むデータ
基本的に mem から送られてくるが初期値設定やシステムコールで siki が介入することがある。
mem_to_R (siki.vhd) => mem_to_R

```
  sys_sig           : out std_logic; --call system call
```

システムコールが呼ばれるというシグナル
ALU, mem を介して siki に渡される。

sys_sig => sys_call_sig (siki.vhd) => sys_call_sig (ALU.vhd)

sys_type : out std_logic_vector(1 downto 0); --system call type
呼ばれたシステムコールの種類
ALU,mem を介して siki に渡される。
sys_type => sys_call_type (siki.vhd) => sys_call_type (ALU.vhd)

branch_instr : out std_logic;
命令が分岐命令であるというシグナル
ALU に渡されて PC の操作を行う。
branch_instr => check_branch (siki.vhd) => check_branch (ALU.vhd)

branch_cond : out std_logic_vector(2 downto 0);
分岐命令の種類を示す。
ALU に渡されて PC の操作の制御を行う。
branch_cond => which_cond (siki.vhd) => which_cond (ALU.vhd)

jump_instr : out std_logic;
命令がジャンプ命令であるというシグナル
ALU に渡されて PC の操作を行う。
jump_instr => check_jump (siki.vhd) => check_jump (ALU.vhd)

store_instr : out std_logic;
命令がストア命令であるというシグナル
ALU を介して mem でデータをストアする。
store_instr => store_sig (siki.vhd) => store_sig (ALU.vhd)

load_instr : out std_logic;
命令がロード命令であるというシグナル。
ALU を介して mem でデータをロードし、id にデータを戻してレジスタに書き込む
load_instr => load_sig (siki.vhd) => load_sig (ALU.vhd)

mem_write : out std_logic; --write data to memory
命令がロード命令であるというシグナル
(不要な可能性あり、というか load_instr と重複していてこれも使われてなかった…)
mem_write => write_mem_sig (siki.vhd) => write_mem_sig (ALU.vhd)

mem_to_R_sig : out std_logic; --write data to register from memory
メモリから読み込んだデータをレジスタに書き込むかを指示するシグナル
ALU,mem を介して id に戻り、レジスタを更新するかを指示する。
mem_to_R_sig => write_R_sig (siki.vhd) => write_R_sig (ALU.vhd)

write_data_R : out std_logic_vector(4 downto 0); --write to this register
メモリから読み込んだデータをどのレジスタに書き込むかを指示する。
ALU,mem を介して id に戻り、どのレジスタを更新するか指定する。

write_data_R => write_R (siki.vhd) => write_R (ALU.vhd)

now_PC : out std_logic_vector(15 downto 0);
現在の PC を表す。
ALU に渡し、次の PC の決定に用いられる。
now_PC => base_PC (siki.vhd) => base_PC (ALU.vhd)

add_to_PC : out std_logic_vector(15 downto 0); --add this to PC
分岐命令等で現在の PC から相対 PC を求める際に now_PC に加える値
add_to_PC => add_PC (siki.vhd) => add_PC (ALU.vhd)

change_PC : out std_logic_vector(15 downto 0); --jump to this PC
ジャンプ命令等で now_PC に代わり次の PC とする値
change_PC => jump_PC (siki.vhd) => jump_PC (ALU.vhd)

data_1 : out std_logic_vector(31 downto 0);
ALU で演算に用いられる引数の 1 番目
data_1 => input_int_1 (siki.vhd) => input_int_1 (ALU.vhd)

data_2 : out std_logic_vector(31 downto 0);
ALU で演算に用いられる引数の 2 番目
data_2 => input_int_2 (siki.vhd) => input_int_2 (ALU.vhd)

data_to_mem : out std_logic_vector(31 downto 0); --store this data
ストア命令でメモリに格納するデータ
ALU を介して mem に渡され、ALU での演算で求められたアドレスに格納される。
data_to_mem => store_data (siki.vhd) => store_data (ALU.vhd)

ALU_control : out std_logic_vector(3 downto 0); --write this
ALU の演算でどのような操作が行われるかを指示する。
ALU_control => calcu_type (siki.vhd) => calcu_type (ALU.vhd)

r0_data : out std_logic_vector(31 downto 0);
r1_data : out std_logic_vector(31 downto 0);
r2_data : out std_logic_vector(31 downto 0);
r3_data : out std_logic_vector(31 downto 0);
r4_data : out std_logic_vector(31 downto 0);
r5_data : out std_logic_vector(31 downto 0);
r6_data : out std_logic_vector(31 downto 0);
r7_data : out std_logic_vector(31 downto 0);
r8_data : out std_logic_vector(31 downto 0);
r9_data : out std_logic_vector(31 downto 0);
r10_data : out std_logic_vector(31 downto 0);

```
  r11_data          : out std_logic_vector(31 downto 0);
  r12_data          : out std_logic_vector(31 downto 0);
  r13_data          : out std_logic_vector(31 downto 0);
  r14_data          : out std_logic_vector(31 downto 0);
  r15_data          : out std_logic_vector(31 downto 0);
  r16_data          : out std_logic_vector(31 downto 0);
  r17_data          : out std_logic_vector(31 downto 0);
  r18_data          : out std_logic_vector(31 downto 0);
  r19_data          : out std_logic_vector(31 downto 0);
  r20_data          : out std_logic_vector(31 downto 0);
  r21_data          : out std_logic_vector(31 downto 0);
  r22_data          : out std_logic_vector(31 downto 0);
  r23_data          : out std_logic_vector(31 downto 0);
  r24_data          : out std_logic_vector(31 downto 0);
  r25_data          : out std_logic_vector(31 downto 0);
  r26_data          : out std_logic_vector(31 downto 0);
  r27_data          : out std_logic_vector(31 downto 0);
  r28_data          : out std_logic_vector(31 downto 0);
  r29_data          : out std_logic_vector(31 downto 0);
  r30_data          : out std_logic_vector(31 downto 0);
  r31_data          : out std_logic_vector(31 downto 0));
```

各レジスタのデータ
siki で読み取れるようにしている。
rX_data => rX_data (siki.vhd) (X ∈ N)

・・Component

(なし)

・・Signal

```
 signal sys_sig_buf      : std_logic := '0';
 signal sys_type_buf     : std_logic_vector(1 downto 0) := "00";
 signal branch_instr_buf : std_logic := '0';
 signal branch_cond_buf  : std_logic_vector(2 downto 0) := "000";
 signal jump_instr_buf   : std_logic := '0';
 signal store_instr_buf  : std_logic := '0';
 signal load_instr_buf   : std_logic := '0';
 signal mem_write_buf    : std_logic := '0';
 signal mem_to_R_sig_buf : std_logic := '0';
 signal write_data_R_buf : std_logic_vector(4 downto 0) := "00000";
 signal now_PC_buf       : std_logic_vector(15 downto 0) := x"0000";
 signal add_to_PC_buf    : std_logic_vector(15 downto 0) := x"0000";
 signal change_PC_buf    : std_logic_vector(15 downto 0) := x"0000";
 signal data_1_buf       : std_logic_vector(31 downto 0) := x"00000000";
```

```
signal data_2_buf       : std_logic_vector(31 downto 0) := x"00000000";
signal data_to_mem_buf  : std_logic_vector(31 downto 0) := x"00000000";
signal ALU_control_buf  : std_logic_vector(3 downto 0) := "0000";
```
バッファ群
state が””11”になったとき ALU に渡している。

```
signal head             : std_logic_vector(5 downto 0) := "000000";
```
命令データの head 部分
命令の種類の特定に用いられる。

```
signal rs_pointer       : std_logic_vector(4 downto 0) := "00000";
```
命令データの rs 部分
rs に格納するレジスタのデータを指定する。

```
signal rt_pointer       : std_logic_vector(4 downto 0) := "00000";
```
命令データの rt 部分
rt に格納するレジスタのデータを指定する。

```
signal rd_pointer       : std_logic_vector(4 downto 0) := "00000";
```
命令データの rd 部分
rd が存在する命令で ALU で求めた値をどのレジスタに格納するかを指定する。

```
signal sa               : std_logic_vector(4 downto 0) := "00000";
```
命令データの sa 部分
小さな即値として ALU の演算で用いられる。

```
signal tail             : std_logic_vector(5 downto 0) := "000000";
```
命令データの tail 部分
命令の種類の特定に用いられる。

```
signal offset           : std_logic_vector(15 downto 0) := x"0000";
```
命令データの offset 部分
PC の操作や ALU の引数として用いられる即値を表す。

```
signal instr_index      : std_logic_vector(25 downto 0) :=
"00000000000000000000000000";
```
命令データの instr_index 部分
大きな即値として ALU の演算で用いられる。

```
signal code             : std_logic_vector(25 downto 6) := x"00000";
```
命令データの code 部分
システムコールで用いられる特別なデータ

signal rs : std_logic_vector(31 downto 0) := x"00000000";
ALU の引数設定で用いられる。

signal rt : std_logic_vector(31 downto 0) := x"00000000";
ALU の引数設定で用いられる。

signal sys_sig_sub : std_logic := '0';
システムコールを実行するかを命令データから判断し、ほかのシグナルの設定に用いられる。

signal sys_type_sub : std_logic_vector(1 downto 0) := "00";
実行するシステムコールの種類をレジスタと mem から戻ってくる書き込みデータから判断

signal branch_instr_sub : std_logic := '0';
分岐命令かを命令データから判断

signal branch_cond_sub : std_logic_vector(2 downto 0) := "000";
分岐命令の種類を命令データから判断

signal jump_instr_sub : std_logic := '0';
ジャンプ命令かを命令データから判断

signal store_instr_sub : std_logic := '0';
ストア命令かを命令データから判断

signal load_instr_sub : std_logic := '0';
ロード命令かを命令データから判断

signal mem_write_sub : std_logic := '0';
ロード命令かを命令データから判断
(不要な可能性あり、というか Port で書いたとおり不要)

signal mem_to_R_sig_sub : std_logic := '0';
メモリからレジスタに書き込むかを命令データから判断する。

signal write_data_R_sub : std_logic_vector(4 downto 0) := "00000";
メモリからどのレジスタに書き込むかを命令データから判断する。

signal now_PC_sub : std_logic_vector(15 downto 0) := x"0000";
現在の PC を格納
(不要な可能性あり、これはシグナルの書き方の統一性のために存在)

signal add_to_PC_sub : std_logic_vector(15 downto 0) := x"0000";
現在の PC に加える値を命令データから判断

signal change_PC_sub : std_logic_vector(15 downto 0) := x"0000";

ジャンプ先の PC を命令データから判断

```
signal data_1_sub       : std_logic_vector(31 downto 0) := x"00000000";
```
ALU に渡す一つ目の引数を命令データから判断

```
signal data_2_sub       : std_logic_vector(31 downto 0) := x"00000000";
```
ALU に渡す二つ目の引数を命令データから判断

```
signal data_to_mem_sub  : std_logic_vector(31 downto 0) := x"00000000";
```
mem に書き込むデータを命令データから判断
ALU を介して mem に渡し、ALU で求めたアドレスに格納する。

```
signal ALU_control_sub  : std_logic_vector(3 downto 0) := "0000";
```
ALU で行う演算の種類を命令データから判断

```
signal r0           : std_logic_vector(31 downto 0) := x"00000000";
signal r1           : std_logic_vector(31 downto 0) := x"00000000";
signal r2           : std_logic_vector(31 downto 0) := x"00000000";
signal r3           : std_logic_vector(31 downto 0) := x"00000000";
signal r4           : std_logic_vector(31 downto 0) := x"00000000";
signal r5           : std_logic_vector(31 downto 0) := x"00000000";
signal r6           : std_logic_vector(31 downto 0) := x"00000000";
signal r7           : std_logic_vector(31 downto 0) := x"00000000";
signal r8           : std_logic_vector(31 downto 0) := x"00000000";
signal r9           : std_logic_vector(31 downto 0) := x"00000000";
signal r10          : std_logic_vector(31 downto 0) := x"00000000";
signal r11          : std_logic_vector(31 downto 0) := x"00000000";
signal r12          : std_logic_vector(31 downto 0) := x"00000000";
signal r13          : std_logic_vector(31 downto 0) := x"00000000";
signal r14          : std_logic_vector(31 downto 0) := x"00000000";
signal r15          : std_logic_vector(31 downto 0) := x"00000000";
signal r16          : std_logic_vector(31 downto 0) := x"00000000";
signal r17          : std_logic_vector(31 downto 0) := x"00000000";
signal r18          : std_logic_vector(31 downto 0) := x"00000000";
signal r19          : std_logic_vector(31 downto 0) := x"00000000";
signal r20          : std_logic_vector(31 downto 0) := x"00000000";
signal r21          : std_logic_vector(31 downto 0) := x"00000000";
signal r22          : std_logic_vector(31 downto 0) := x"00000000";
signal r23          : std_logic_vector(31 downto 0) := x"00000000";
signal r24          : std_logic_vector(31 downto 0) := x"00000000";
signal r25          : std_logic_vector(31 downto 0) := x"00000000";
signal r26          : std_logic_vector(31 downto 0) := x"00000000";
```

```
signal r27          : std_logic_vector(31 downto 0) := x"00000000";
signal r28          : std_logic_vector(31 downto 0) := x"00000000";
signal r29          : std_logic_vector(31 downto 0) := x"ffffffff"; --$sp
signal r30          : std_logic_vector(31 downto 0) := x"00010000"; --$hp
signal r31          : std_logic_vector(31 downto 0) := x"00000000"; --$ra
```

レジスタの実態
ここに各レジスタの値が格納されている。

```
signal r0_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r1_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r2_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r3_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r4_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r5_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r6_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r7_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r8_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r9_sub       : std_logic_vector(31 downto 0) := x"00000000";
signal r10_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r11_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r12_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r13_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r14_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r15_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r16_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r17_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r18_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r19_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r20_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r21_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r22_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r23_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r24_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r25_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r26_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r27_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r28_sub      : std_logic_vector(31 downto 0) := x"00000000";
signal r29_sub      : std_logic_vector(31 downto 0) := x"00010000";
signal r30_sub      : std_logic_vector(31 downto 0) := x"ffffffff";
signal r31_sub      : std_logic_vector(31 downto 0) := x"00000000";
```

レジスタの値を更新する際に用いられる。
siki から送られる mem_to_R_sig_return が'1' のとき外部から送られるデータにレジスタを更新する。

·ALU.vhd (ALU)

· ·Port