ICS EXE 4

2022.03.16

In original architecture, the stack (memory) is used to keep the **return address**. However, in this problem, we directly use a register to keep the return address for a better performance. Suppose two new instructions, namely **`rcall`** and **`rret`**, are used to replace **`call`** and **`ret`** instruction in Y86 instruction sets, which have the following encoding. (NOTE the original call and ret instruction will never be used)

			Byte	0		1		2	10
`rcall`	`rB`	`valC`		`E`	`Fn`	`F`	`rb`	`valc`	
`rret`	`rB`			`E`	`Fn`	`F`	`rb`		

1. Please fill in the **generic** function of each stage for **`rcall rB valC`** and **`rret rB`** on updated **sequential** implementation like Figure 4.21.
   (NOTE: fill all functions in each stage, and use '–' for empty stage)

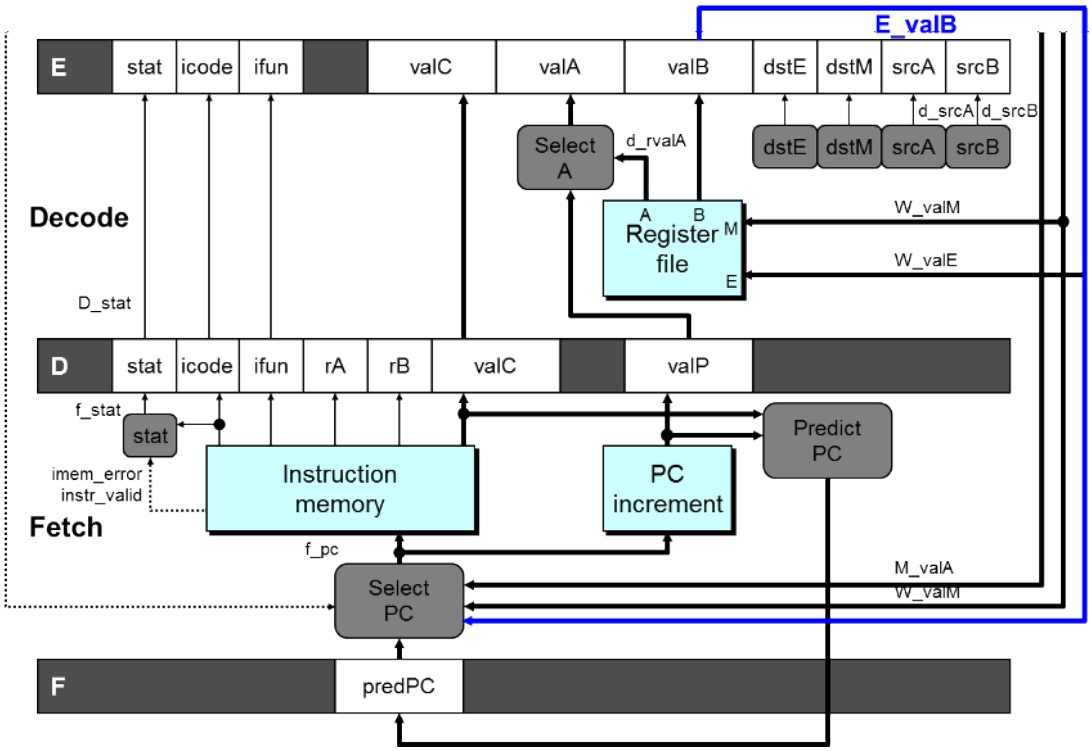
Field	rcall rB valC	rret rB
Fetch		
Decode		
Execute		
Memory		
Write Back		
PC update		

2. Suppose we just add a new forwarding logic from **`W_valE`** to **`f_pc`**, and the rest of pipeline hardware structure is the same to original (Figure 4.41). Please describe all possible hazards due to the two new instructs respectively. You need provide detail explanation and list **detection conditions** like Figure 4.64 and **control action** like Figure 4.66.

Condition	Trigger

Condition	Pipeline register				
	F	D	E	M	W

3. As shown in the following new PIPE- logic figure, we add a **return forwarding** logic from **`E_valB`** to **`f_pc`** to take back return address for the new instructions. Please describe all possible hazards again on optimized hardware structure. You still need provide detail explanation and list **detection conditions** like Figure 4.64 and **control action** like Figure 4.66.



4. Now we add **return forwarding** (mentioned in problem 3) to PIPE like Figure 4.52, please describes the modification and provides increased HCL code of **`f_pc`**, ,**`F_stall`**, **`D_stall`** and **`D_bubble`** logic for two new instructions (**`rcall`** and **`rret`**). (NOTE: you need to provide all increased codes due to **`rcall`** and **`rret`**, even the symbol **`IRCALL`** or **`IRRET`** **don't appear in the expression directly**)

   For example:

   ```
   bool instr_valid = f_icode in {IRCALL , IRRET};
   ```

5. Compared with the original instruction set and hardware structure (Figure 4.52), the two new instructions (**`rcall`** and **`rret`**) and return forwarding will cause new combinations of hazards. Please draw the **pipeline states** figure (Figure 4.67) and list **pipeline control action** (see the table of Problem 4.37 and 4.38) for new combinations about new instructions.

6. Please calculate the number of **cycles** and **waste cycles** for the following codes in **original** and new architecture base on PIPE (figure 4.52) with return forwarding. The initial value of all registers are **zero** and we always use **TAKEN** branch prediction strategy for all conditional jump. (**Hint**: you need calculate the number of cycles until the last stage of the last instruction)

Original

```
    irmovq    Stack,%rsp
    irmovq    $2,%rax
loop:
    call      foo
    irmovq    $-1,%rbx
    addq      %rbx,%rax
    jne       loop
    halt
foo:
    ret
    irmovq    $1,%rax
Stack:
```

Cyclyes:

Wasted cycles:

New

```
    irmovq    Stack,%rsp
    irmovq    $2,%rax
loop:
    rcall     %rdi,foo
    irmovq    $-1,%rbx
    addq      %rbx,%rax
    jne       loop
    halt
foo:
    rret      %rdi
    irmovq    $1,%rax
Stack:
```

Cyclyes:

Wasted cycles: