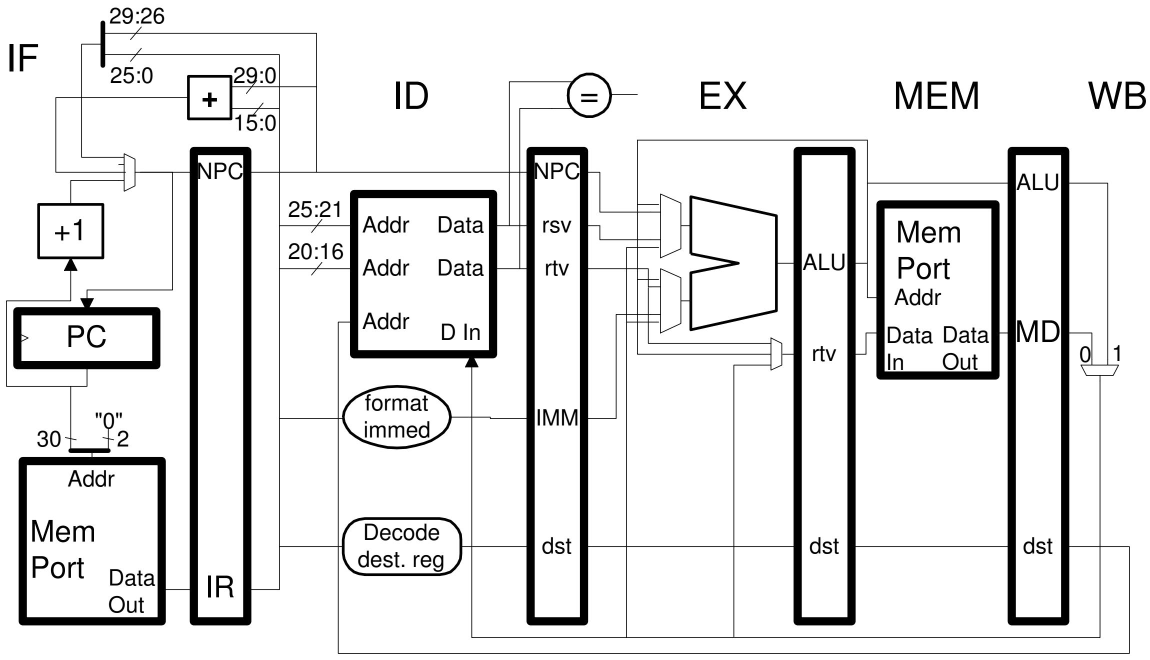
LSU EE 4720 **Homework 4** Solution **Due: 7 November 2005**

Problem 1: The code below executes on the implementation illustrated.

(*a*) Draw a pipeline execution diagram up until the first fetch of the third iteration.

(*b*) What is the CPI for a large number of iterations?



```
# Solution:
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
addi r3, $0, 123    IF ID EX ME WB
LOOP:
lw r1, 0(r2)           IF ID EX ME WB
bne r1, r3, LOOP          IF ID ----> EX ME WB
lw r2, 4(r1)                 IF ----> ID EX ME WB
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
LOOP:
lw r1, 0(r2)                          IF ID -> EX ME WB
bne r1, r3, LOOP                         IF -> ID ----> EX ME WB
lw r2, 4(r1)                                   IF ----> ID EX ME WB
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
lw r1, 0(r2)                                            IF ...
```

In the first iteration `lw r1, 0(r2)` executes without a stall but in the second iteration it stalls in ID and so the first iteration cannot be used to compute CPI. The second and third iterations start with the processor in the same state (the branch in EX and the second load in ID, see cycles 6 and 12), and so the second iteration can be used to compute the CPI. $\boxed{\text{The CPI is } \frac{12-6}{3} = \frac{6}{3} = 2.}$

Problem 2: Is there any way to add bypass paths to the implementation above so that the code executes with fewer stalls:

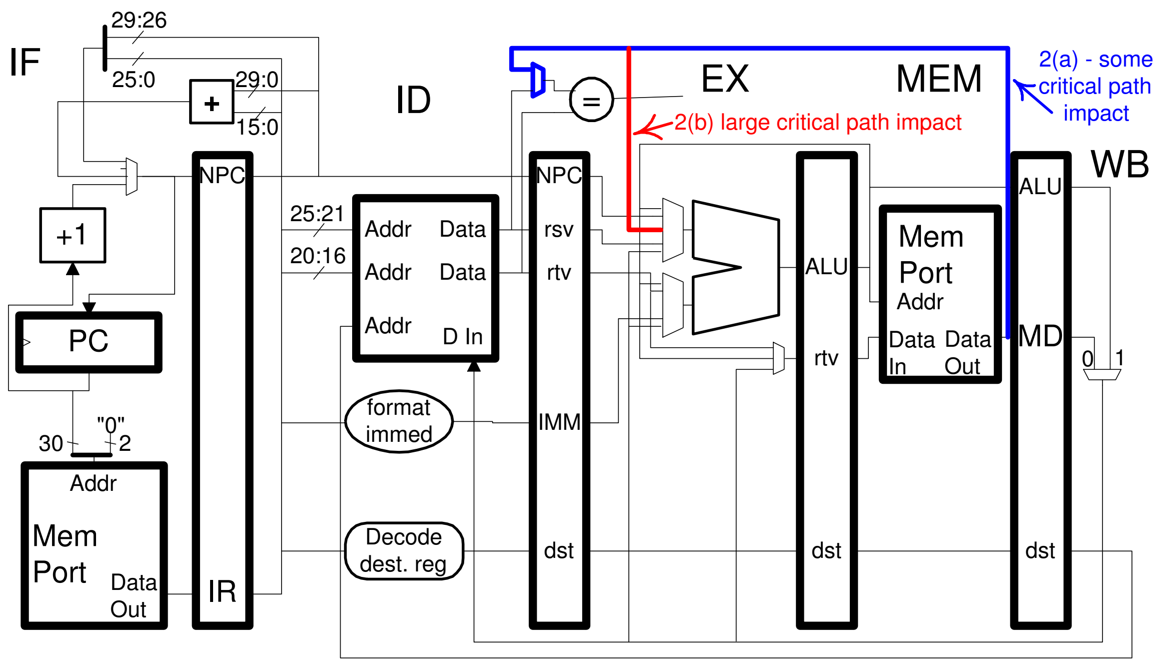
(*a*) Suggest bypass paths that might have critical path impact but which probably won't halve the clock frequency.

To avoid the branch stall bypass the value from the output of the MEM-stage memory port to the comparator in ID, shown in blue below. This will probably impact critical path because the memory port is probably using the whole cycle. It won't halve the clock frequency because the comparison can be done quickly (certainly less than a cycle because it ordinarily waits for the register file).

(*b*) Explain why it is impossible to remove all stalls by adding bypass paths.

A bypass path for the `lw r2, 4(r1)` to `lw r1, 0(r2)` dependence would go from the output of the memory port to the ALU input (shown in red below), each of those devices uses most of its clock cycle and so clock frequency would be halved. That's really bad and one should never do it, but it's not impossible.

A bypass to handle the `lw r1, 0(r2)` to `bne r1, r3, LOOP` dependence *is* impossible because the branch needs the loaded value one cycle before its available. For example, in the solution the branch needs the loaded value in cycle 3, but the load instruction has not yet reached MEM.



Problem 3: The `beqir` instruction from the midterm exam solution compares the contents of the `rs` register to the immediate, if the two are equal the branch is taken, the address of the branch target is in the `rt` register. In the code example below `beqir` compares the contents of `r3` to the constant 123, if they are equal the branch is taken with register `r1` holding the target address, in this case to `TARG`. The delay slot, `nop`, is also executed.

(*a*) Show the changes needed to implement this instruction on the implementation above.

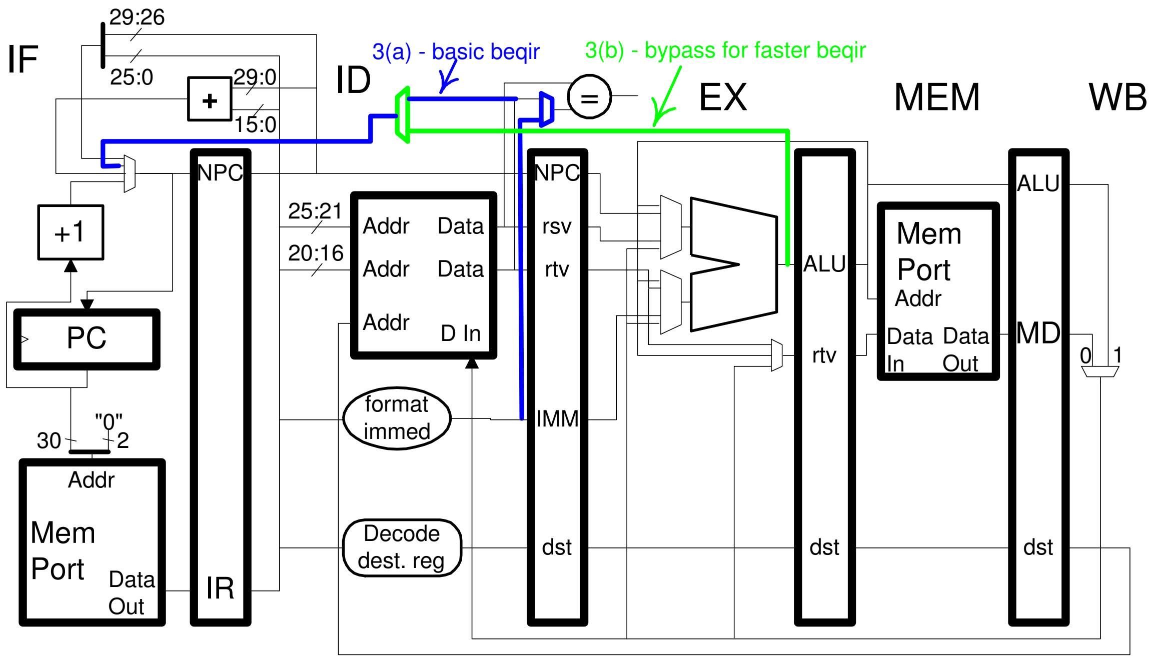
Changes shown below in blue. Two changes were made. First, a multiplexer is put before comparison unit in ID to select either the rt register value (regular branches) and the immediate (`beqir`). Second, the rt register value is sent to the PC mux in IF.

(*b*) Include bypass paths so that the code below executes as fast as possible:

```
 lui r1, hi(TARG)
 ori r1, r1, lo(TARG)
 beqir r3, 123,  r1
 nop

 # Lots more code.
TARG:
 xor r9, r10, r11
```

The code above has a dependence from `ori` to `beqir`. To bypass the value a bypass path was added from EX to ID, shown in green. This bypass path may stretch the critical path because two multiplexers have been added to the path at the output of the ALU.



IF
ID
EX
MEM
WB
3(a) - basic beqir
3(b) - bypass for faster beqir
29:26
25:0
29:0
15:0
+
=
NPC
IR
+1
PC
"0"
30
2
Addr
Mem
Port
Data
Out
25:21
20:16
Addr
Data
Addr
Data
Addr
D In
format immed
Decode dest. reg
NPC
rsv
rtv
IMM
dst
ALU
rtv
dst
Mem Port
Addr
Data In
Data Out
ALU
MD
dst
0
1