**EE 4720** **Homework 2** **Due:** $t$,

$$\text{where } t = \begin{cases} \text{4 October 1999,} & \text{if 29 September class held;} \\ \text{6 October 1999,} & \text{if 29 September class cancelled;} \\ \text{8 October 1999,} & \text{if 29 September and 1 October class cancelled.} \end{cases}$$

**Problem 1:** Suppose the coding of DLX instructions were changed so the destination appeared before the source operands, as shown in the codings below:

New Type R:

| Opcode | rd | rs1 | rs2 | func |
|---|---|---|---|---|
| 0 | | | | |
| 0 5 | 6 10 | 11 15 | 16 20 | 21 31 |

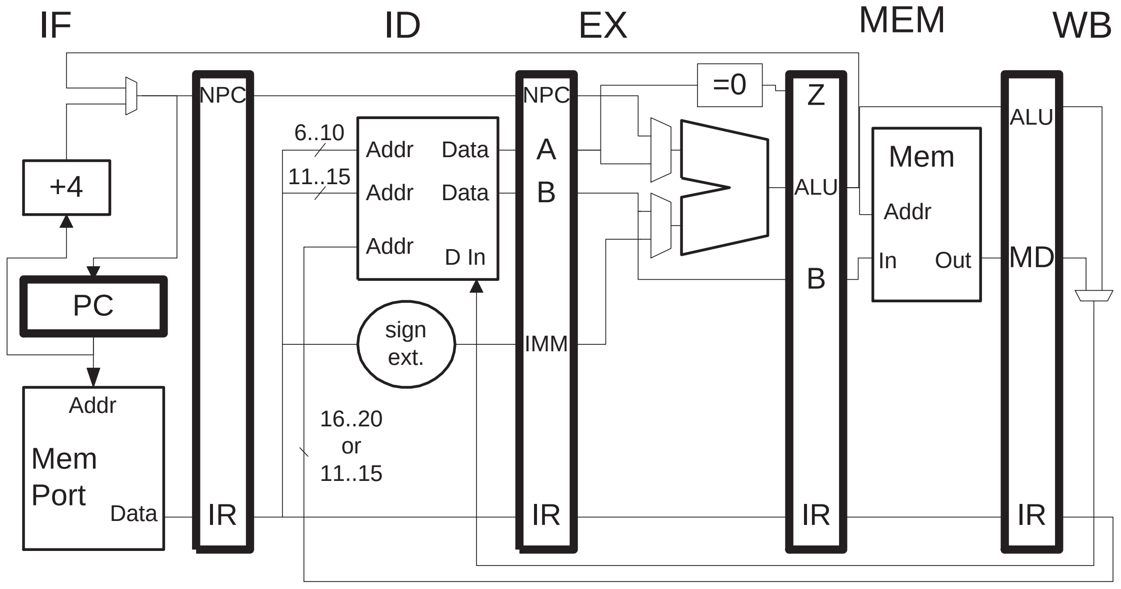
New Type I:

| Opcode | rd | rs1 | Immediate |
|---|---|---|---|
| | | | |
| 0 5 | 6 10 | 11 15 | 16 31 |

Type J: (no change)

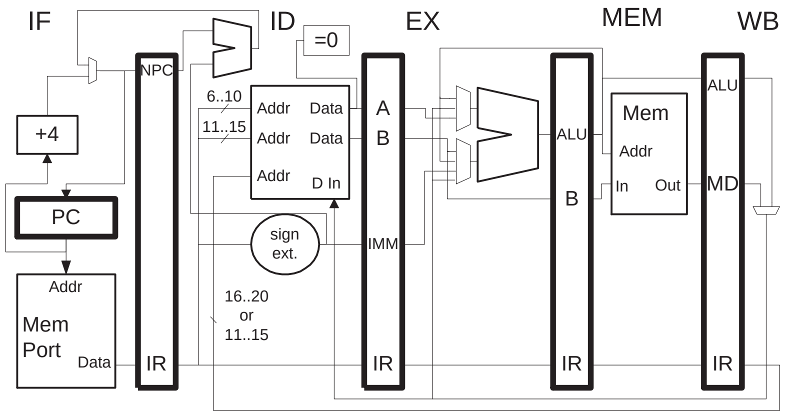
| Opcode | Offset |
|---|---|
| | |
| 0 5 | 6 31 |

Show the changes needed to the pipeline below to implement this new ISA. The changes should only effect the ID and WB stages. If there are differences in the control inputs to multiplexors or other units, explain what those differences are. Make sure that your design executes store instructions correctly.



**Problem 2:** The program below executes on the DLX implementation shown below. The implementation uses forwarding (bypassing) to avoid some data hazards and stalls to avoid others. All forwarding paths are shown. (If a needed forwarding path is not there, sorry, you'll have to stall.) A value can be read from the register file in the same cycle it is written. The destination field in the `beqz` is zero. Instructions are nulled (squashed) in this problem by replacing them with `slt r0,r0,r0`. All instructions stall in the ID stage.

```
 ! Initially, r1=0x101, r2=0x202, r3=0x303
 ! MEM[0x103] = 0xfe
 sub  r0, r0, r0
 sub  r0, r0, r0
 sub  r0, r0, r0
 sub  r0, r0, r0
 sub  r0, r0, r0
START: ! START = 0x50
 lb   r1, 2(r1)
 addi r1, r1, #3
 or   r1, r1, r2
 beqz r2, SKIP !(taken)
 add  r3, r1, r2
 sub  r0, r0, r0
 sub  r0, r0, r0
SKIP:
 xor  r3, r1, r3
 sub  r0, r0, r0
 sub  r0, r0, r0
 sub  r0, r0, r0
```



The table below shows the contents of pipeline registers and changes to architecturally visible registers `r1-r31` over time. Cycle zero is the time that `lb` is in instruction fetch. The first two columns are completed; fill in the rest of the table. Use a "?" for the value of the "immediate field" of a type R instruction and for the output of the memory when no memory read is performed. Show pipeline register values even if they're not used. Assume that the ALU performs the branch target computation even though it was already computed in ID. The row labeled "Reg. Chng." shows a new register value that is available at the *beginning* of the cycle. If no register value is written leave the entry blank.

*Hints: See Spring 1999 HW 3 for a similar problem. One feature of the solution would not be present if* `lb` *were replaced by a* `addi`*. Another feature may not be present if* `lb` *were replaced by* `lw`*.*

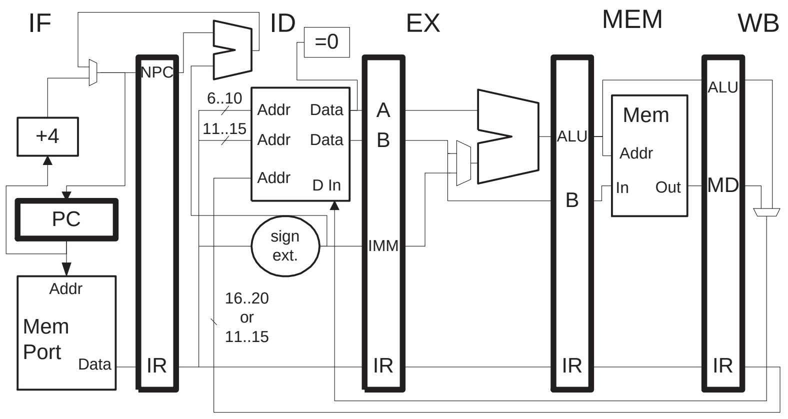
| Cycle | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| PC | 0x50 | 0x54 | | | | | | | | | |
| IF/ID.IR | sub | lb | | | | | | | | | |
| Reg. Chng. | r0 ←0 | r0 ←0 | | | | | | | | | |
| ID/EX.IR | sub | sub | | | | | | | | | |
| ID/EX.A | 0 | 0 | | | | | | | | | |
| ID/EX.B | 0 | 0 | | | | | | | | | |
| ID/EX.IMM | ? | ? | | | | | | | | | |
| EX/MEM.IR | sub | sub | | | | | | | | | |
| EX/MEM.ALU | 0 | 0 | | | | | | | | | |
| EX/MEM.B | 0 | 0 | | | | | | | | | |
| MEM/WB.IR | sub | sub | | | | | | | | | |
| MEM/WB.ALU | 0 | 0 | | | | | | | | | |
| MEM/WB.MD | ? | ? | | | | | | | | | |

**Problem 3:** Consider the program:

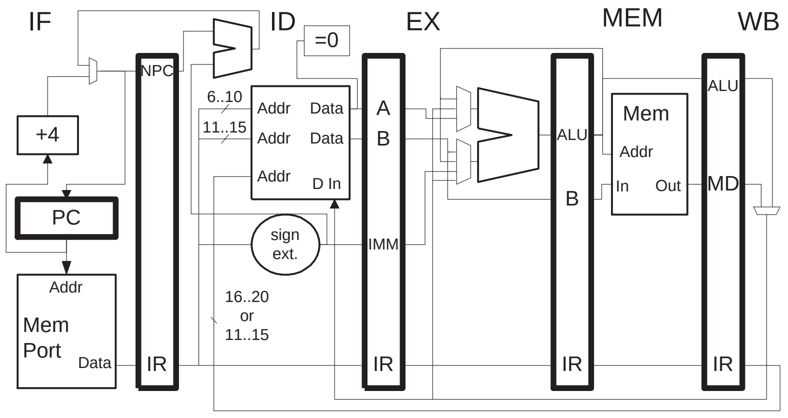
```
LOOP:
 lw   r1, 0(r2)
 add  r3, r1, r3
 addi r2, r2, #4
 bneq r1, LOOP
 or   r4, r5, r6
```

For each implementation below provide a pipeline execution diagram showing execution up to the third fetch of `lw` and determine the CPI for a large number of iterations.

Not Bypassed:

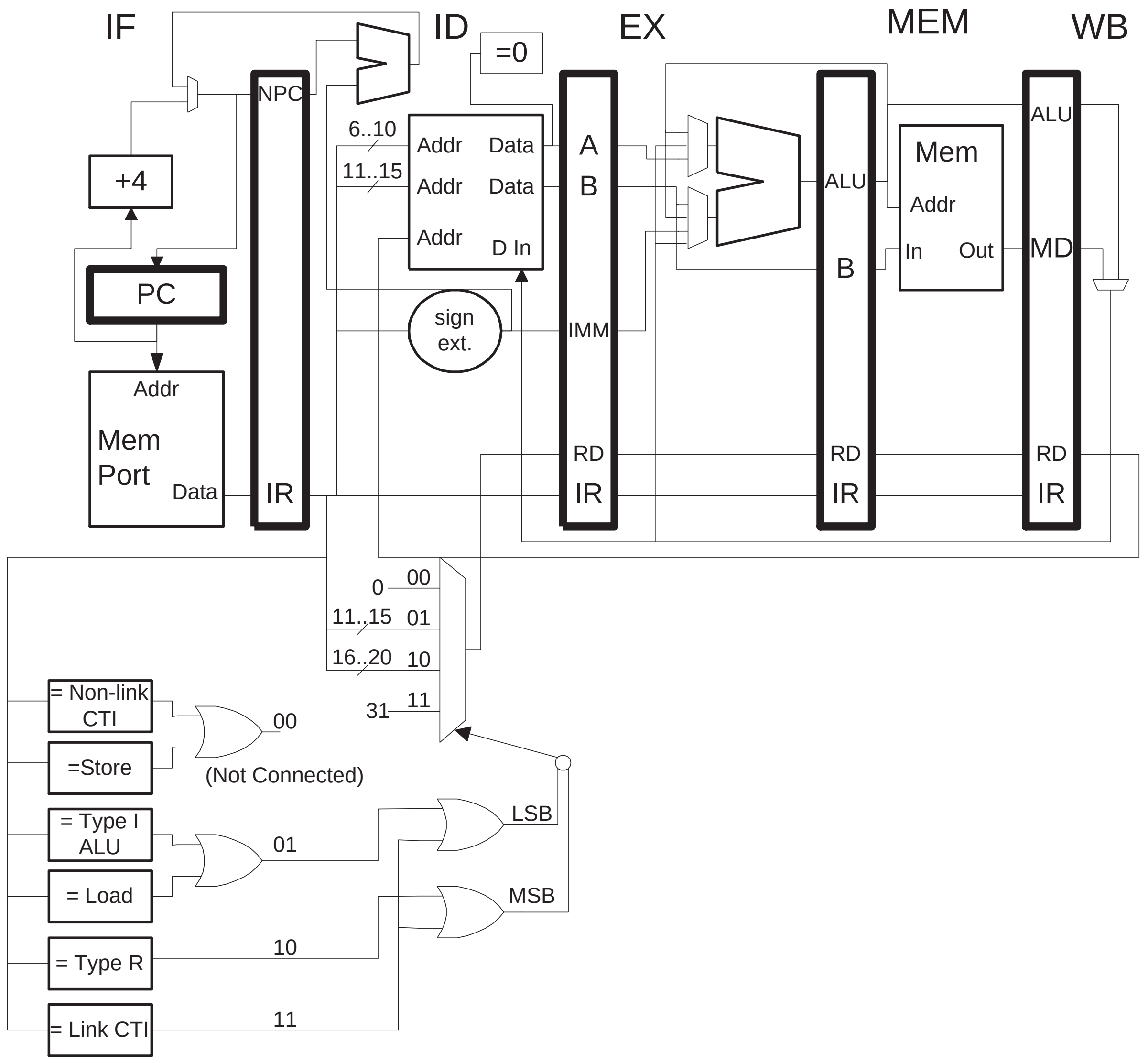


Bypassed:



**Problem 4:** Schedule (rearrange) the instructions in the program used in the previous problem to improve execution speed. (Do not change what the program does!). Show pipeline execution diagrams and determine CPI for the two implementations.

**Problem 5:** Show the changes needed to implement the predicated instructions presented in class. (Set 4, page 25, as of this writing.) Describe the instruction format and show any datapath and control changes to the implementation below.


IF
ID
EX
MEM
WB
=0
NPC
+4
PC
Addr
Mem Port
Data
IR
6..10
11..15
Addr
Data
Addr
Data
Addr
D In
A
B
IMM
sign ext.
RD
IR
ALU
B
RD
IR
Mem
Addr
In
Out
ALU
MD
RD
IR
0
00
11..15
01
16..20
10
31
11
= Non-link CTI
=Store
00
(Not Connected)
= Type I ALU
= Load
01
LSB
MSB
= Type R
10
= Link CTI
11