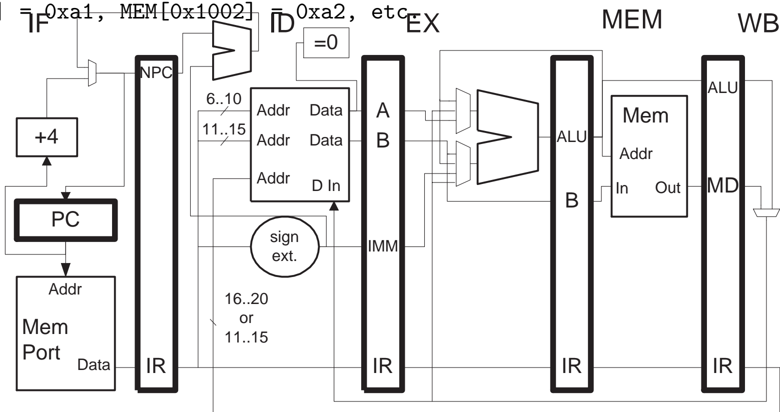
EE 4720 Homework 2 Solution Due: 23 February 2000

Problem 1: 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 with `slt r0,r0,r0`. All instructions stall in the ID stage.

```
 ! Initially, r1=0x1000, r2=0x2000, r3=0x3000
 ! MEM[0x1000] = 0xa0, MEM[0x1001] = 0xa1, MEM[0x1002] = 0xa2, etc.
 sub  r0, r0, r0
 sub  r0, r0, r0
 sub  r0, r0, r0
 sub  r0, r0, r0
START: ! START = 0x50
 addi r1, r1, #8
 lh   r2, 2(r1)
 sw   4(r1), r2
 bneq r2, START  (taken)
 sub  r2, r3, r1
 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 `addi` 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 `r0` is written leave the entry blank.

Hint: See Spring 1999 HW 3 and Fall 1999 HW 2 for similar problems.

Completed table appears below. Numbers in table are in hexadecimal.

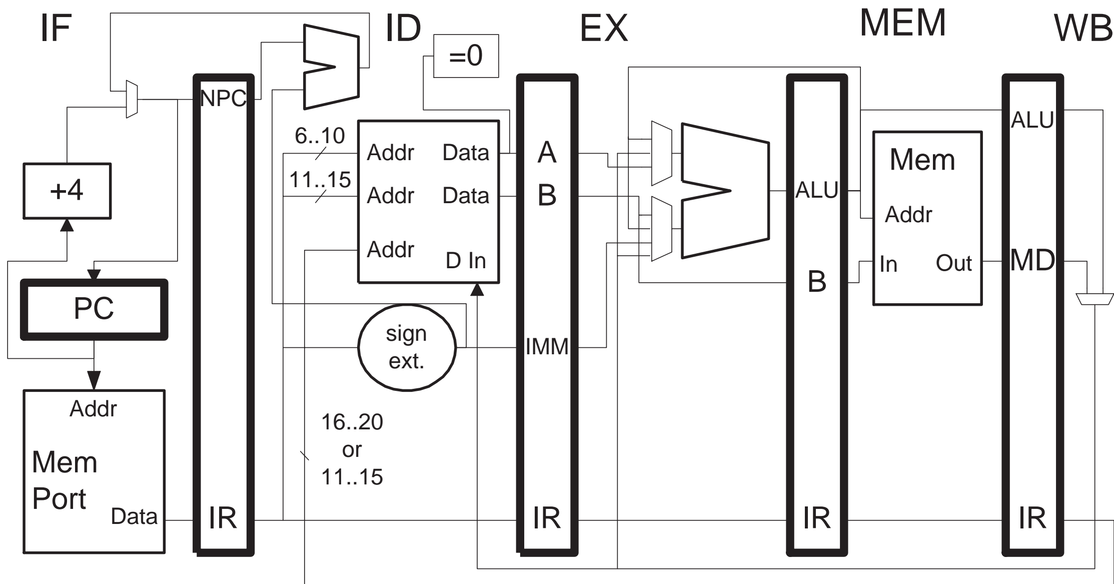
Cycle	0	1	2	3	4	5	6	7	8	9	10
PC	50	54	58	5c	5c	5c	60	50	54	58	5c
IF/ID.IR	sub	addi	lh	sw	sw	sw	bneq	sub	addi	lh	sw
Reg. Chng.	r0 ←0	r0 ←0	r0 ←0	r0 ←0	r1 ←1008	r2 ←ffffaaab	r0 ←0	r0 ←0	r0 ←0	r0 ←0	r0 ←0
ID/EX.IR	sub	sub	addi	lh	slt	slt	sw	bneq	sub	addi	lh
ID/EX.A	0	0	1000	1000	1000	1008	1008	ffffaaab	3000	1008	1008
ID/EX.B	0	0	1000	2000	2000	2000	ffffaaab	0	1008	1008	ffffaaab
ID/EX.IMM	?	?	8	2	4	4	4	-4	?	8	2
EX/MEM.IR	sub	sub	sub	addi	lh	slt	slt	sw	bneq	sub	addi
EX/MEM.ALU	0	0	0	1008	100a	1	1	100c	14	1ff8	1010
EX/MEM.B	0	0	0	1000	2000	2000	2000	ffffaaab	0	1008	1008
MEM/WB.IR	sub	sub	sub	sub	addi	lh	slt	slt	sw	bneq	sub
MEM/WB.ALU	0	0	0	0	1008	100a	1	1	100c	14	1ff8
MEM/WB.MD	?	?	0	0	?	ffffaaab	?	?	?	?	?

To help solve the problem, find a pipeline execution diagram for the code (shown below). Cycle numbers in diagram and table match.

```
LOOP:
!                  0    1    2    3    4    5    6    7    8    9    10
addi r1, r1, #8   IF   ID   EX   MEM  WB             IF   ID   EX   MEM
lh   r2, 2(r1)         IF   ID   EX   MEM  WB             IF   ID   EX
sw   4(r1), r2              IF   ID   ----->  EX   MEM  WB   IF   ID
bneq r2, LOOP                    IF   ----->  ID   EX   MEM  WB   IF
sub  r2, r3, r1                               IFx
}
```

Problem 2: The execution of the code in the problem above should suffer a stall (not including the branch delay). Add bypass path(s) to the diagram below needed to avoid the stall(s). Add **only** the bypass paths needed to avoid the stalls encountered in the problem above, and **no others**. (The diagram below is the same as the one in the first problem.)

```
START:
addi r1, r1, #8
lh   r2, 2(r1)
sw   4(r1), r2
bneq r2, START
sub  r2, r3, r1
```



Add a bypass path from the output of the WB-stage multiplexor to a new multiplexor feeding the MEM-stage memory data-in port. (The other input to the new multiplexor is from `EX/MEM.B`.)