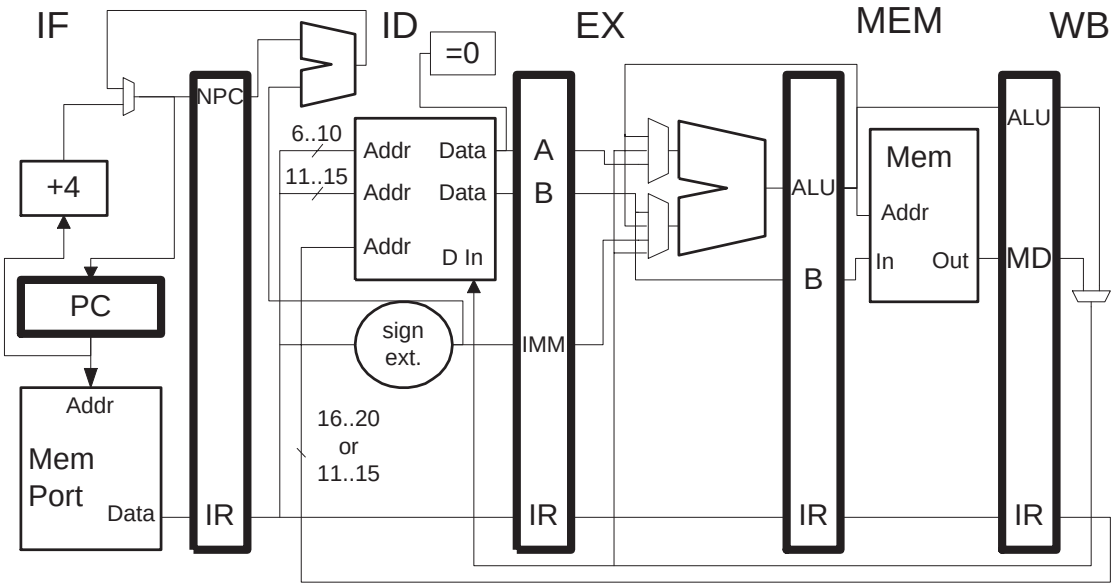
EE 4720 **Homework 3** **Due: 2 October 2000**

Problem 1: What changes would have to be made to the pipeline below to add the DLX-BAM indexed addressing instructions (from homework 2). *Hint: The load is easy and inexpensive, the store requires a substantial change.* Add the changes to the diagram below, but omit the control logic. Do explain how the control logic would have to be changed.

```
! Indexed addressing.
lw r1, (r2+r3)  ! r1 = MEM[ r2 + r3 ];
sw (r2+r3), r4  !  MEM[ r2 + r3 ] + r4;
```



Problem 2: *For maximum pedagogical benefit solve the problem above before attempting this one.* The integer pipeline of the Sun Microsystems microSPARC-IIep implementation of the SPARC V8 ISA is similar to the Chapter-3 implementation of DLX that is being covered in class.

What are the stage names and abbreviations used in the microSPARC-IIep? *Hint: This is really easy once you've found the right page.*

SPARC V8 includes indexed addressing, for example:

```
ld [%o3+%o0], %o2   ! Load word: %o2 = MEM[ %o3 + %o0 ]
st %o0, [%o1+%g1]   ! Store word: MEM[ %o1 + %g1 ] = %o0
```

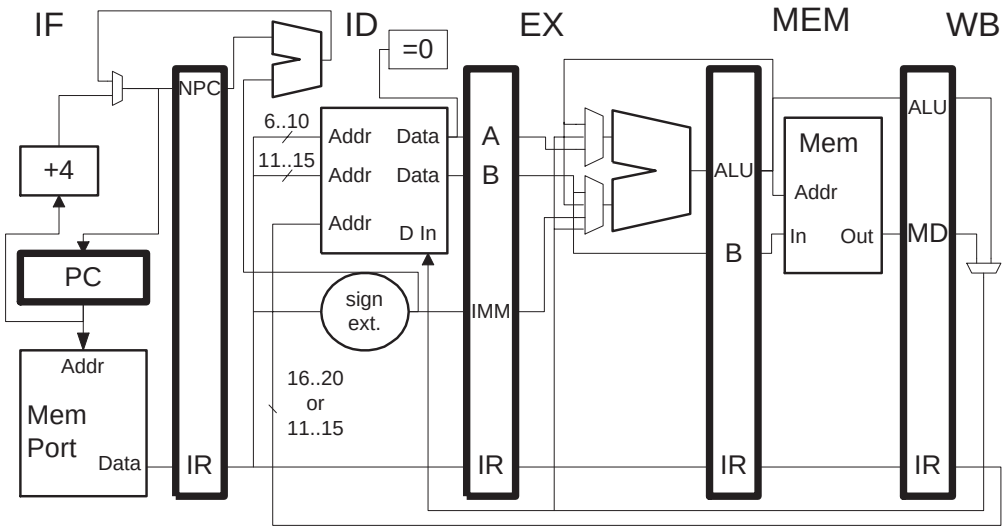
(Register `%o0` is a real register, *not* a special zero register.) What are the differences between the microSPARC-IIep integer pipeline and the Chapter-3 DLX pipeline that allow it to execute an indexed store? Be sure to answer the question directly, do not copy or paraphrase **irrelevant** material. A shorter answer is preferred.

Information on the microSPARC-IIep can be found via
`http://www.sun.com/microelectronics/manuals/microSPARC-IIep/802-7100-01.pdf`
or `http://www.ee.lsu.edu/ee4720/microsparc-IIep.pdf`. Those who enjoy a challenge can study the diagram on page 10, however the material to answer the question can be found early in Chapter 3. The manual uses many terms which have not yet been covered in class, the question can still be answered once the right page is found. The manual is 256 pages so don't print the whole thing.

Problem 3: The following pipeline execution diagram shows the execution of a program on the DLX implementation shown below. The implementation uses forwarding (bypassing) to avoid some data hazards and stalls to avoid others; connections needed to implement the `jalr` instruction are not shown. A value can be read from the register file in the same cycle it is written. Instructions are squashed (nulled) in this problem by replacing them with `or r0,r0,r0`. All instructions stall in the ID stage.

Add the datapath connections needed so the `jalr` executes as shown.

```
 ! Initially, r1=0x100, r2=0x200, r3=0x300, r4 = 0x68
 ! The lw will read 0xaaa0.
 ! Cycle              0  1  2  3  4  5  6  7  8  9  10
 sub  r0, r0, r0      WB
 sub  r0, r0, r0      ME WB
 sub  r0, r0, r0      EX ME WB
 sub  r0, r0, r0      ID EX ME WB
START: ! START = 0x50
 add r2, r2, r3       IF ID EX ME WB
 lw  r2,4(r2)            IF ID EX ME WB
 sw  8(r2), r1              IF ID -> EX WE WB
 jalr r4                       IF -> ID EX ME WB
 xor r4, r1, r2                      IFx
 subi  r2, r1, #0x10
 andi  r2, r2, #0x20                    IF ID EX ME WB
 slti  r3, r3, #0x30                       IF ID EX ME
```



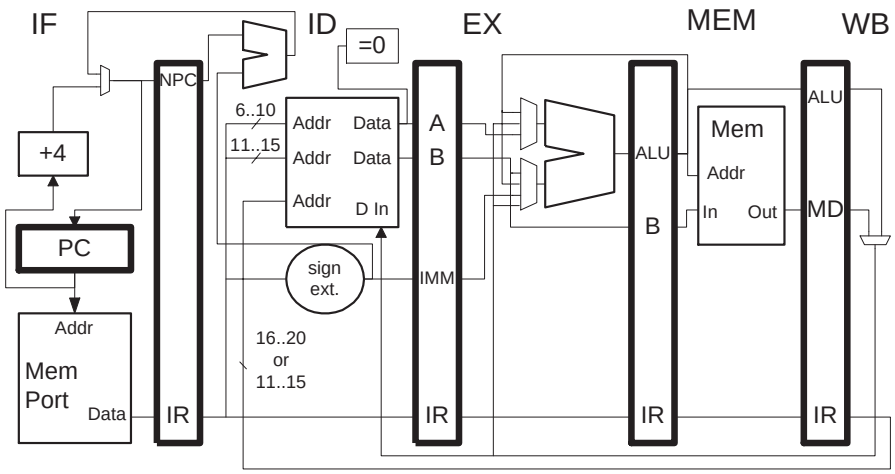
The table on the next page shows the contents of pipeline registers and changes to architecturally visible registers `r1-r31` over time. The first two columns are completed; fill in the rest of the table. Use a "?" for the value of the "immediate field" of a type R instruction and for the output of the memory when no memory read is performed. Show pipeline register values even if they're not used. 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: For hints and confirmation see Spring 1999 HW 3, Fall 1999 HW 2, and Spring 2000 HW 2, linked to `http://www.ee.lsu.edu/ee4720/prev.html`*, for similar problems. It's important that the problem is solved by inspection of the diagram,* **not** *by inferring mindless, unworthy-of-an-engineer rules from past solutions. Mindless rules are hard to remember and are useless in new situations.*

Cycle	0	1	2	3	4	5	6	7	8	9
PC	50	54								
IF/ID.IR	sub	add								
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 4: Draw a pipeline execution diagram showing the execution of the familiar code below until the second fetch of `lw` (the beginning of the second iteration). *Hint: There are RAW hazards associated with the loads, stores, and the branch.* What is the CPI for a large number of iterations?

```
LOOP:
 lw r1, 0(r2)
 add r3, r1, r4
 lb  r5, 0(r3)
 sb  0(r6), r5
 addi r2, r2, #4
 addi r6, r6, #1
 slt r7, r2, r8
 bneq r7, LOOP
 xor r10, r11, r12
```



Problem 5: Rearrange (schedule) the instructions in the program from the previous problem to minimize the number of stalls. *Now* what is the CPI for a large number of iterations? *Hint: The offsets in the load and store instructions can be changed, even to negative numbers.*