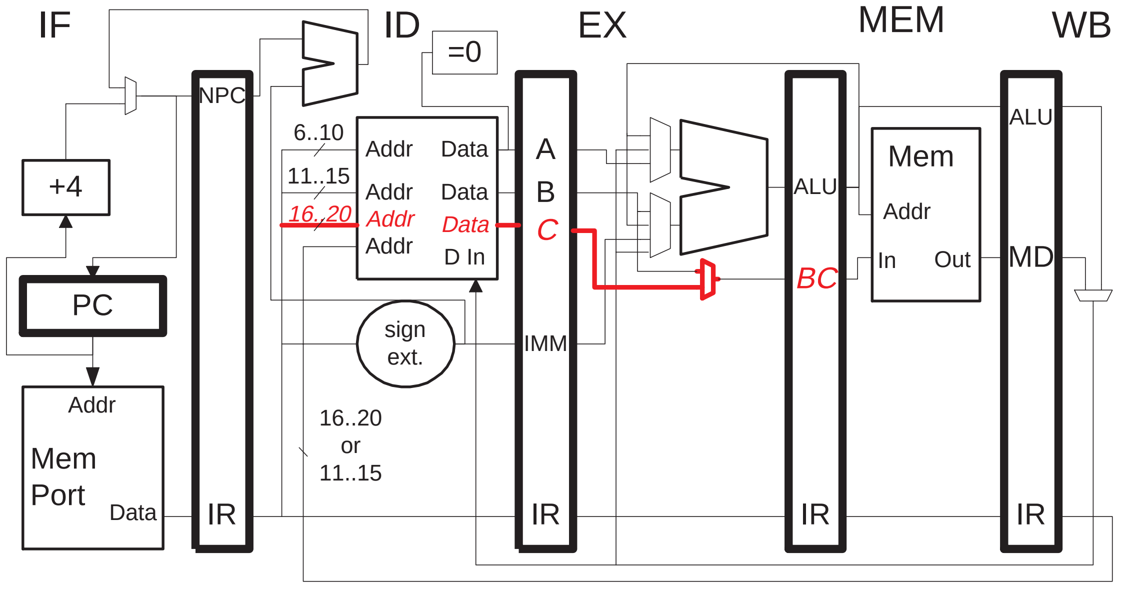
EE 4720 **Homework 3** Solution **Due: 2 October 2000**

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;
```

No datapath changes are needed to implement the indexed load. The control logic must recognize the new instruction type and use the `A` and `B` inputs to the ALU rather than the `A` and `IMM` that are used for ordinary loads.

The changes needed to implement the indexed store are shown in red bold below. A third read port is added to the register file in `ID` and a multiplexor is added to route either the `ID/EX.B` or the new `ID/EX.C` latch to the memory data in in the `MEM` stage. Control logic changes are similar to the indexed load, with the addition of control for the new multiplexor.



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.

The register file has a third read port, used for store data. The store data is read in the E stage, rather than in D, as it would in the DLX implementation.

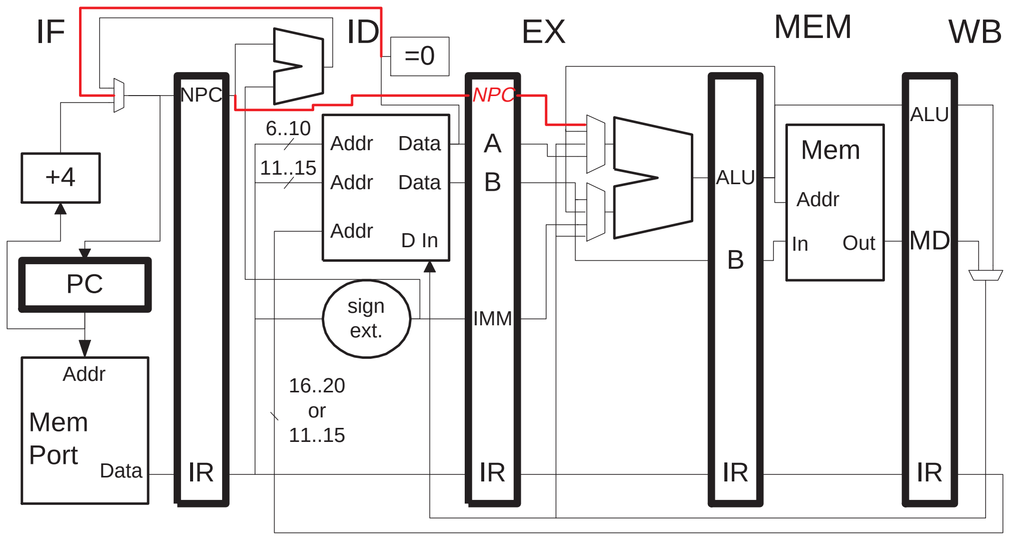
Problem 3: The following pipeline execution diagram shows the execution of a program on the DLX implementation shown below. The implementation uses forwarding (bypassing) to avoid some data hazards and stalls to avoid others; connections needed to implement the `jalr` instruction are not shown. A value 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.

Instruction addresses are shown below, to the left of the instructions.

```
        ! 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
0x40    sub  r0, r0, r0      WB
0x44    sub  r0, r0, r0      ME WB
0x48    sub  r0, r0, r0      EX ME WB
0x4c    sub  r0, r0, r0      ID EX ME WB
      START: ! START = 0x50
0x50    add r2, r2, r3       IF ID EX ME WB
0x54    lw  r2,4(r2)            IF ID EX ME WB
0x58    sw  8(r2), r1              IF ID -> EX WE WB
0x5c    jalr r4                       IF -> ID EX ME WB
0x60    xor r4, r1, r2                      IFx
0x64    subi  r2, r1, #0x10
0x68    andi  r2, r2, #0x20                    IF ID EX ME WB
0x6c    slti  r3, r3, #0x30                       IF ID EX ME
```

Changes for `jalr` are show below in red bold. For the `jalr` instruction the ALU will pass through the top input unchanged. As an alternative, `EX/MEM.NPC` and `MEM/WB.NPC` registers could be included, with the output of `MEM/WB.NPC` going into the same multiplexor as `MEM/WB.ALU` and `MEM/WB.MD`. This would require two more registers, but those `NPC` registers might be needed for exception processing.



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.

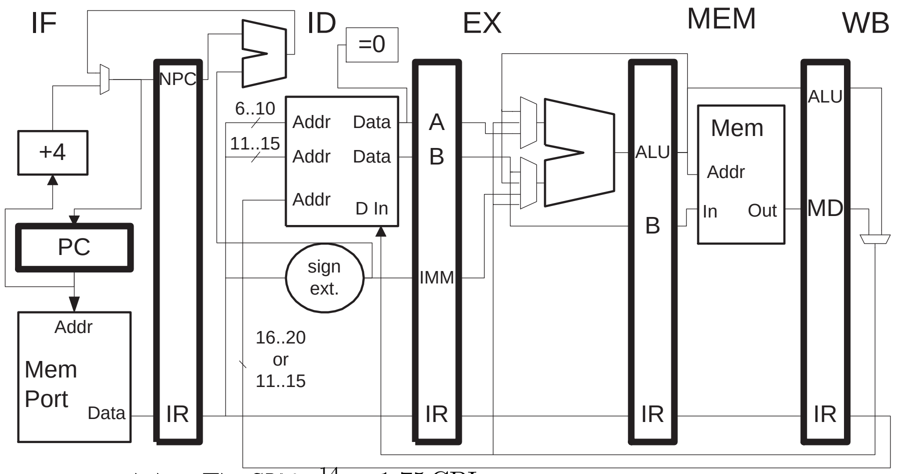
Hint: For hints and confirmation see Spring 1999 HW 3, Fall 1999 HW 2, and Spring 2000 HW 2, linked to `http://www.ee.lsu.edu/ee4720/prev.html`*, for similar problems. It's important that the problem is solved by inspection of the diagram,* **not** *by inferring mindless, unworthy-of-an-engineer rules from past solutions. Mindless rules are hard to remember and are useless in new situations.*

Completed table appears below. The numbers in the table are in hexadecimal. An "x" after an instruction name indicates it has been squashed.

Cycle	0	1	2	3	4	5	6	7	8	9
`PC`	50	54	58	5c	5c	60	68	6c		
`IF/ID.IR`	sub	add	lw	sw	sw	jalr	xor	andi	slti	
Reg. Chng.	`r0` ←0	`r0` ←0	`r0` ←0	`r0` ←0	`r2` ←500	`r2` ←aaa0	`r0` ←0	`r0` ←0	`r31` ←60	`r0` ←0
`ID/EX.IR`	sub	sub	add	lw	swx	sw	jalr	xorx	andi	slti
`ID/EX.A`	0	0	200	200	200	500	68	0	aaa0	300
`ID/EX.B`	0	0	300	200	100	100	?	0	aaa0	300
`ID/EX.IMM`	?	?	?	4	8	8	?	?	20	30
`EX/MEM.IR`	sub	sub	sub	add	lw	swx	sw	jalr	xorx	andi
`EX/MEM.ALU`	0	0	0	500	504	300	aaa8	60	0	20
`EX/MEM.B`	0	0	0	300	200	100	100	?	0	aaa0
`MEM/WB.IR`	sub	sub	sub	sub	add	lw	swx	sw	jalr	xorx
`MEM/WB.ALU`	0	0	0	0	500	504	300	aaa8	60	0
`MEM/WB.MD`	?	?	?	?	?	aaa0	?	?	?	?

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
```



The pipeline execution diagram appears below. The CPI is $\frac{14}{8} = 1.75\,\mathrm{CPI}$.

```
! Solution.
LOOP:
 ! Cycle            0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17
 lw r1, 0(r2)       IF ID EX ME WB                            IF ID EX ME
 add r3, r1, r4        IF ID -> EX ME WB                         IF ID ->
 lb  r5, 0(r3)            IF -> ID EX ME WB                         IF ->
 sb  0(r6), r5                  IF ID ----> EX ME WB
 addi r2, r2, #4                   IF ----> ID EX ME WB
 addi r6, r6, #1                            IF ID EX ME WB
 slt r7, r2, r8                                IF ID EX ME WB
 bneq r7, LOOP                                    IF ID ----> EX ME WB
 xor r10, r11, r12                                   IF ----> x
```

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.*

The pipeline execution diagram appears below. The CPI is $\frac{9}{8} = 1.25\,\mathrm{CPI}$.

```
LOOP:
 ! Cycle          0  1  2  3  4  5  6  7  8  9  10 11
 lw r1, 0(r2)     IF ID EX ME WB             IF ID EX
 addi r2, r2, #4     IF ID EX ME WB             IF ID
 add r3, r1, r4         IF ID EX ME WB             IF
 lb  r5, 0(r3)             IF ID EX ME WB
 slt r7, r2, r8               IF ID EX ME WB
 addi r6, r6, #1                 IF ID EX ME WB
 sb  -1(r6), r5                     IF ID EX ME WB
 bneq r7, LOOP                         IF ID EX ME WB
 xor r10, r11, r12                        IF
```