Name ______________________

Computer Architecture
EE 4720
Midterm Examination, Part I

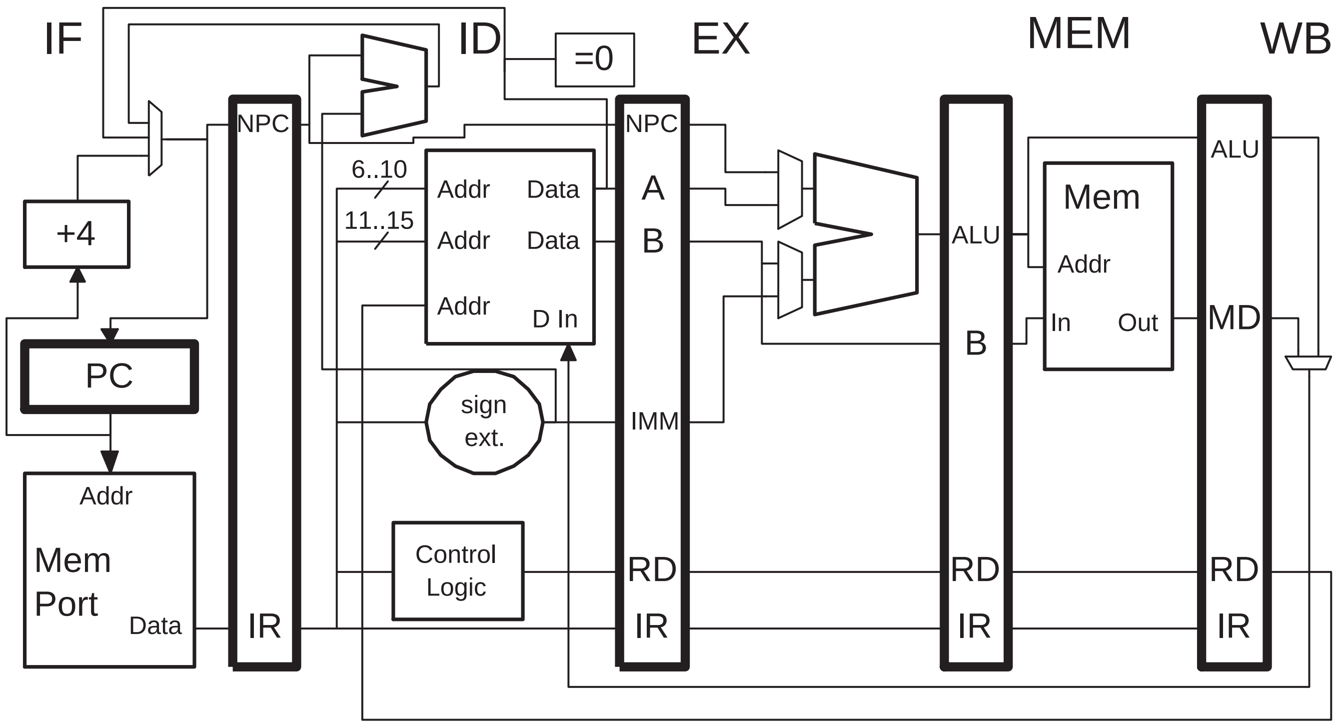
Monday, 16 October 2000, 12:40–13:30 CDT

Problem 1	______	(17 pts) Mon.
Problem 2	______	(17 pts) Mon.
Problem 3	______	(16 pts) Mon.
Problem 4	______	(13 pts) Wed.
Problem 5	______	(17 pts) Wed.
Problem 6	______	(20 pts) Wed.
Exam Total	______	(100 pts)

Alias ______________________

Good Luck!

Problem 1: The program below executes on the pipeline below as illustrated in the pipeline execution diagram below. Bypass paths do not appear in the illustration (below).

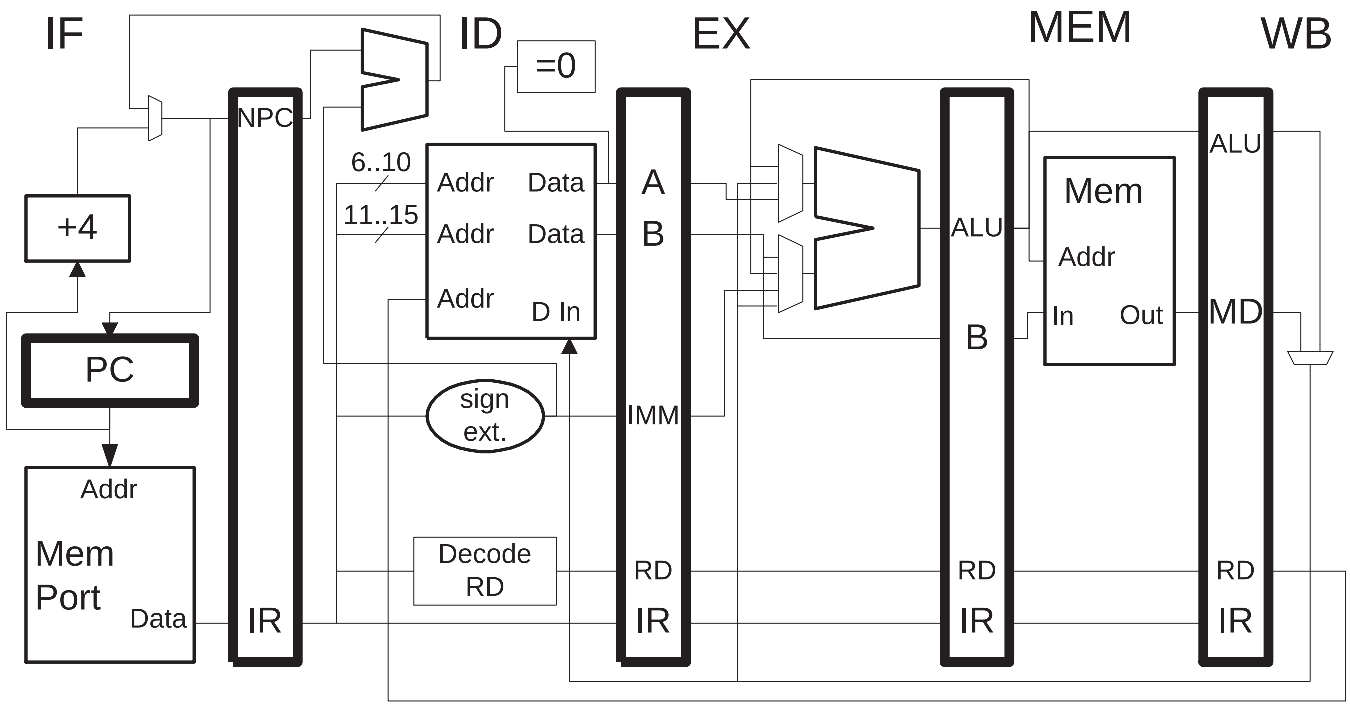



```
! Cycle             0  1  2  3  4  5  6  7  8  9  10 11
andi r8, r8, #31   IF ID EX ME WB
add  r10, r9, r8      IF ID EX ME WB
bnez r8, LINEX           IF ID EX ME WB
jalr r10                    IF ID EX ME WB
xor                            IFx
...
subi r31, r31, #8                 IF ID EX ME WB
sw   0(r10), r31                     IF ID EX ME WB
! Cycle             0  1  2  3  4  5  6  7  8  9  10 11
```

☐ [10 pts] Add *exactly* those bypass paths that are needed so that the code (above) executes as shown. Credit will be deducted for unneeded bypasses. *Please, please, please check the code carefully for dependencies.*

☐ [7 pts] Next to each bypass path indicate the cycle(s) in which it will be used.

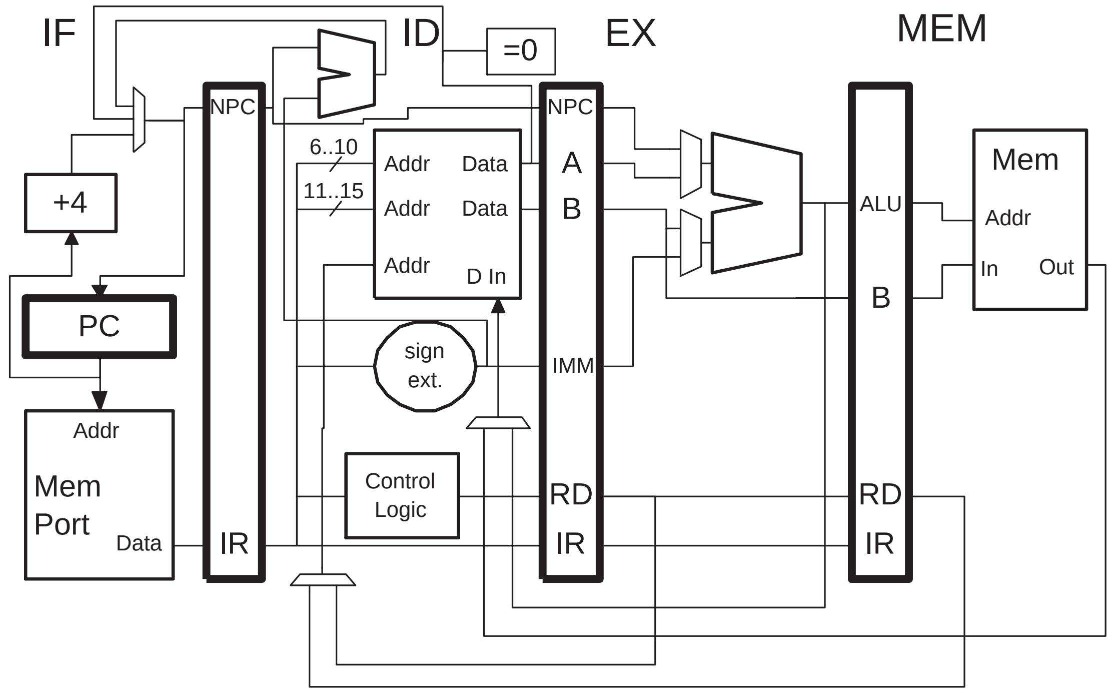
Problem 2: As described in class, postincrement instruction `lw r1,(r2+)` loads the value at memory address `r2` into register `r1` and stores `r2+4` in `r2`. Postincrement stores are similar. The pipeline below is to be modified so that it can execute postincrement loads and stores for bytes, half words, and words. A logic block size can be used; its input is the `opcode` and `func` fields; the output is 0 for a postincrement with a byte-size load or store value, 1 for a postincrement with a half-word value, 2 for a postincrement with a word-size value, and 3 if the instruction is not a postincrement load or store.



☐ [10 pts] Show the datapath changes needed so that the pipeline can execute postincrement store instructions. Include the control logic for obtaining the amount to increment the address by but do not include any other control logic.

☐ [7 pts] Show the additional datapath changes needed so that the pipeline can execute postincrement loads. These changes must not add structural hazards and the load must execute without stalling the pipeline (assuming favorable dependencies).

Problem 3: The four-stage (no WB) pipeline below includes an *Express Writeback* feature, eliminating the need for bypass connections. Instructions proceed through the pipeline slightly differently than the DLX pipeline presented in class. **Do not** add or assume the presence of bypass connections.



☐ [6 pts] Show a pipeline execution diagram for the code below on this pipeline. (Don't forget to look for dependencies.) State any assumptions about how the register file operates.

```
add r1, r2, r3

lw  r4, 8(r1)

sw  12(r5), r4
```

One disadvantage of *Express Writeback* is that it introduces a structural hazard.

☐ [6 pts] Show a program that encounters the hazard and a pipeline execution diagram showing how the hazard can be avoided by stalling. (*Hint: the program can be just two instructions.*)

☐ [4 pts] Explain how *Express Writeback* affects the critical path.

Part II on Wednesday at 12:40 precisely.

Name ____________________________

Computer Architecture
EE 4720
Midterm Examination, Part II

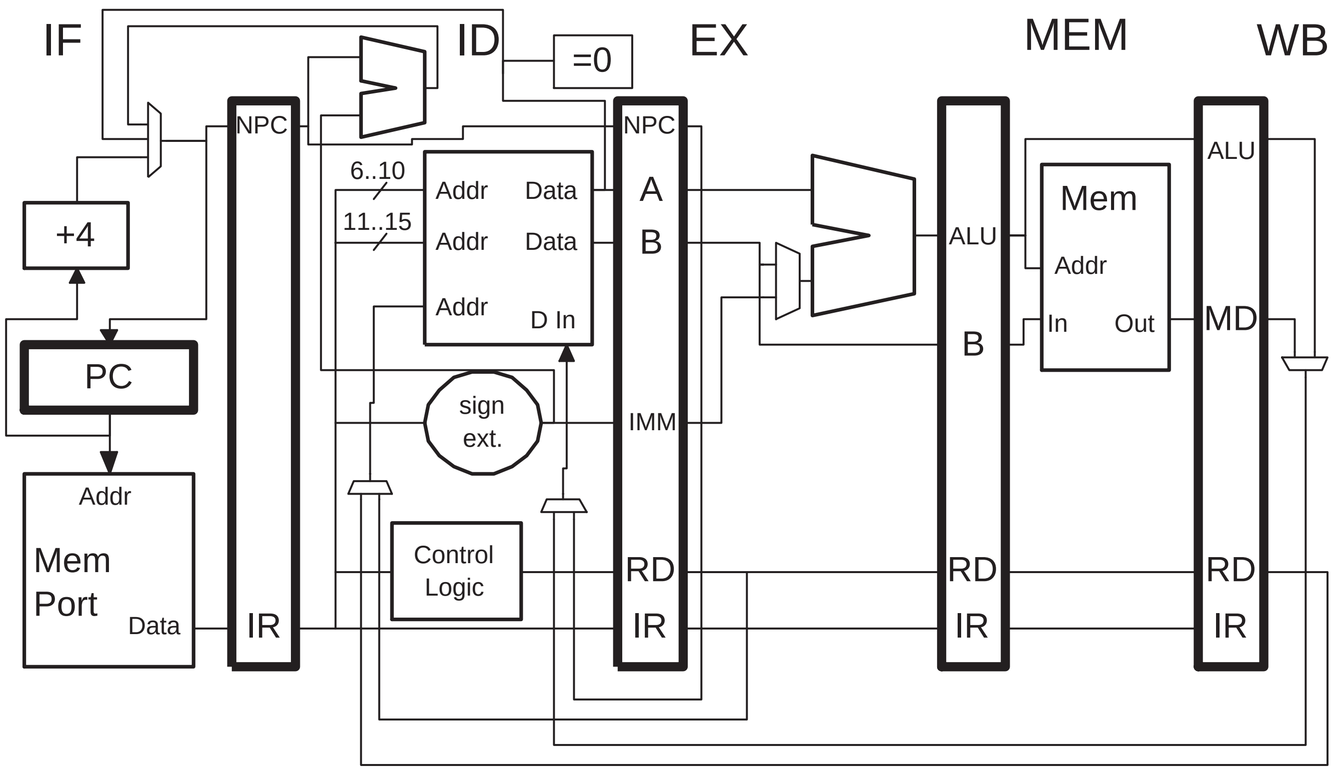
Wednesday, 18 October 2000, 12:40–13:30 CDT

Problem 1	(17 pts)	Mon.
Problem 2	(17 pts)	Mon.
Problem 3	(16 pts)	Mon.
Problem 4	(13 pts)	Wed.
Problem 5	(17 pts)	Wed.
Problem 6	(20 pts)	Wed.
Exam Total	(100 pts)	

Alias ____________________________

Good Luck!

Problem 4: The diagram below includes naïve hardware for implementing the `jal` and `jalr` instructions. With this hardware precise exceptions are impossible.



☐ [6 pts] Explain why precise exceptions are impossible here.

☐ [7 pts] Show a program including a `jalr` that encounters an exception and which does not run correctly (after the exception handler returns) because of the way `jalr` is implemented. Briefly explain why it does not run correctly. (For partial credit answer the question using an instruction other than `jalr`.)

Problem 5: For some, 16 bits is just not enough. Consider a new DLX instruction, `mbi` (move big immediate), which moves a large immediate into register `r1`. (Register `r1` is always used, a different register cannot be specified.) The following code uses the new instruction:

```
mbi  0x12345     ! Move 0x12345 into register r1.
add  r2, r1, r2  ! Use 0x12345 in an add.
```

☐ [5 pts] Describe how the instruction could be coded using an *existing* DLX instruction type to get the biggest immediate possible. Specify the size of the immediate.

Ignoring the previous part, suppose one wanted the immediate to be 30 bits.

☐ [4 pts] Why are 30-bit immediates impossible using an existing instruction type?

☐ [4 pts] Describe how a 30-bit immediate `mbi` could be coded using a *new* DLX instruction type. (See the next subpart before answering.)

Whether it is possible to add a 30-bit immediate instruction and maintain compatibility depends on certain details of DLX which during lectures were usually made up on the spot.

☐ [4 pts] What kind of details were those? Make them up so that the new instruction is compatible.

Problem 6: Answer each question below.

(*a*) Loads and stores in DLX are aligned.

☐ [6 pts] What does that mean? Provide examples of aligned and unaligned accesses.

(*b*) As described in class, an ISA implementing a time data type might have instructions to determine the number of days between two times. (Say between 18 October 2000 and 15 December 2000.)

☐ [7 pts] Give two reasons why an ISA should *not* include such a time data type.

(*c*) RISC and CISC are sometimes seen as two competing architectural styles.

☐ [7 pts] Name two features that distinguish RISC processors from CISC processors.