LSU EE 4720 Homework 5 Solution Due: 4 April 2003

Problem 1: [Easy] Complete pipeline execution diagrams for the following code fragments running on the fully bypassed MIPS implementations with floating point units as described below.

```
# Solution
# One ADD unit, latency 3, initiation interval 1.
add.d f0, f2, f4    IF ID A1 A2 A3 A4 WF
sub.d f6, f0, f8       IF ID -------> A1 A2 A3 A4 WF
add.d f8, f10, f12        IF -------> ID A1 A2 A3 A4 WF

# One ADD unit, latency 3, initiation interval 2.
add.d f0, f2, f4    IF ID A1 A1 A2 A2 WF
sub.d f6, f0, f8       IF ID -------> A1 A1 A2 A2 WF
add.d f8, f10, f12        IF -------> ID -> A1 A1 A2 A2 WF

# Two ADD units (A and B), latency 3, initiation interval 4.
add.d f0, f2, f4    IF ID A  A  A  A  WF
sub.d f6, f0, f8       IF ID -------> A  A  A  A  WF
add.d f8, f10, f12        IF -------> ID B  B  B  B  WF
```

Problem 2: [Easy] Choose the latency and initiation interval for the add and multiply functional units so that the second instruction stalls to avoid a structural hazard. Show a pipeline execution diagram with this execution. (The easy way to solve it is to do the PED first, then figure out the latency and initiation interval.)

```
mul.d f0, f2, f4
add.d f6, f8, f10
```

Both functional units have an initiation interval of 1. The multiply unit has a latency of 3 and the add unit has a latency of 2, so if it were not for the stall they would encounter a structural hazard (the two instructions trying to write their results at the same time).

```
# Solution
mul.d f0, f2, f4   IF ID M1 M2 M3 M4 WF
add.d f6, f8, f10     IF ID -> A1 A2 A3 WF
```

Problem 3: The two PEDs below show execution of MIPS code that produces wrong answers. For each explain why and show a PED of correct execution.

```
# PED showing a DESIGN FLAW. (The code runs incorrectly.)
# Cycle             0  1  2  3  4  5  6  7
add.s f1, f10, f11  IF ID A1 A2 A3 A4 WF
sub.d f2, f0, f4       IF ID A1 A2 A3 A4 WF

# PED showing a DESIGN FLAW. (The code runs incorrectly.)
# Cycle             0  1  2  3  4  5  6  7  8
mul.d f0, f2, f4    IF ID M1 M2 M3 M4 M5 M6 WF
sub.s f1, f10, f11     IF ID A1 A2 A3 A4 WF
```

In both cases the problem is due to the fact that double-precision instructions (`sub.d` and `mul.d` here) actually read and write registers in pairs. The `sub.d`, for example, reads `f0` and `f1` as the first operand (32 bits from each register), `f4` and `f5` as the second operand, and write the result in registers `f2` and `f3`.

The first code fragment does not run correctly because the **sub.d** read **f1** in cycle 1, that is before it is written by the proceeding instruction, in cycle 6. (Note that using instructions this way is unusual, but they still must execute correctly.)

In the second code fragment the **mul.d** overwrites, in cycle 8, the result written by **sub.s** in cycle 7.

```
# Solution.  (Runs correctly assuming a very complete set of bypass paths.)
add.s f1, f10, f11  IF ID A1 A2 A3 A4 WF
sub.d f2, f0, f4       IF ID -------> A1 A2 A3 A4 WF

# Solution.  (Runs correctly.)
mul.d f0, f2, f4    IF ID M1 M2 M3 M4 M5 M6 WF
sub.s f1, f10, f11     IF ID ----> A1 A2 A3 A4 WF
```

Problem 4: As directed to below, design the logic for the floating-point register file in the MIPS implementation illustrated below. The FP portion shows only part of add functional unit. Assume that is the only functional unit.

- Describe how the FP register file works. For reference, here is a description of the integer register file: The integer register file has two read ports and a write port. Each read port has a five-bit address input and a 32-bit data output. The write port has a five-bit address input and a 32-bit data input. Reads from zero retrieve 0, writes to zero have no effect.

- The following signals are available: $\boxed{\text{is dbl}}$; if 1 the instruction uses double-precision operands, otherwise single-precision. $\boxed{\text{FP dst}}$: if 1 the instruction writes the floating-point register file, otherwise it does not (possibly because it's not a floating-point instruction).

- Show all connections to the FP register file. Show the number of bits or the bit range for each connection.

- The WF stage provides two signals, FPU (the value to write back) and fd, something generated in ID (as part of the solution). Additional signals can be sent down the pipeline.

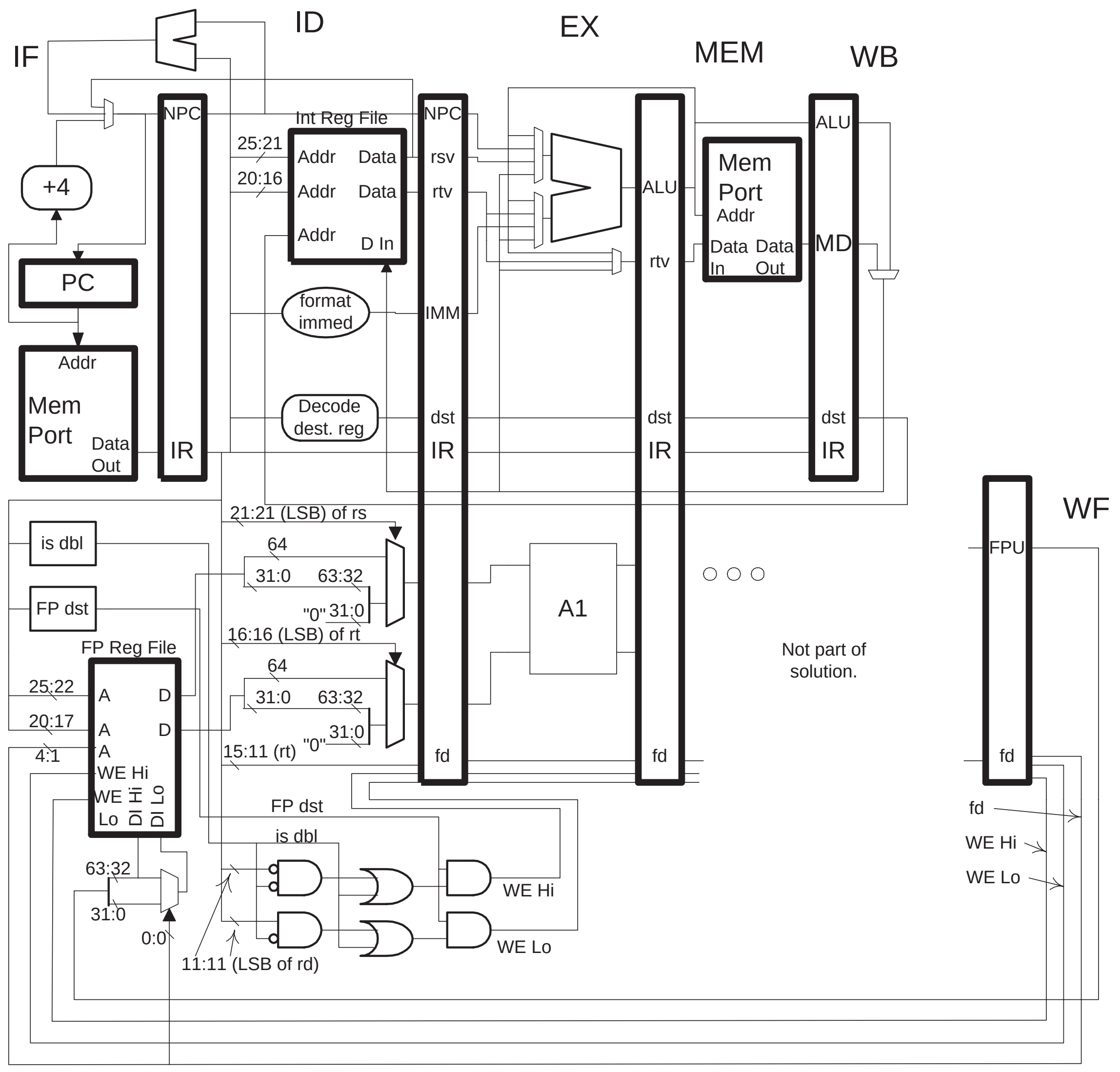
- Keep In Mind: The hardware should work for both single and double operands. (That's what makes the problem interesting. If you're confused first solve it assuming only double operands, then attempt the full problem.)

- Make sure the fragments from the previous problem would run correctly.

Solution shown below. It is assumed that the functional units have 64-bit inputs. If they perform 32-bit operations then they operate on the high bits, bits 63:32.

The register file stores 16 64-bit numbers, each 64-bit number is two registers, say `f0` and `f1`. Notice that the address inputs use just four bits, omitting the LSB of the register number. The outputs of the register file are 64 bits, a multiplexor selects the full 64 bits if the register number is even (LSB 0) or it moves the low 32 bits to the high 32 bits if the register number is odd.

The register file uses a write-enable (WE) signal to control register writes. This was not needed in the integer register file because register zero could be used if nothing was to be written. There are actually two write enable signals, for the high 32 bits (63:32) and for the low bits (31:0). If a double operand is written then both write enables are asserted. If a single is written and the register is even WE high is asserted, otherwise WE low is asserted. The write enable signals are computed in ID and sent down the pipeline to be used in the WF stage.


IF
ID
EX
MEM
WB
WF
NPC
Int Reg File
25:21
20:16
Addr
Data
Addr
Data
Addr
D In
NPC
rsv
rtv
IMM
dst
IR
format immed
Decode dest. reg
+4
PC
Addr
Mem Port
Data Out
IR
ALU
rtv
dst
IR
Mem Port
Addr
Data In
Data Out
ALU
MD
dst
IR
21:21 (LSB) of rs
64
31:0
63:32
"0"
31:0
16:16 (LSB) of rt
64
31:0
63:32
"0"
31:0
15:11 (rt)
A1
fd
fd
is dbl
FP dst
FP Reg File
25:22
20:17
4:1
A
A
A
D
D
WE Hi
WE Lo
DI Hi
DI Lo
63:32
31:0
0:0
FP dst
is dbl
WE Hi
WE Lo
11:11 (LSB of rd)
Not part of solution.
FPU
fd
fd
WE Hi
WE Lo