Advanced Computer Architecture CMSC 611
Homework 3

Due in class Oct 17th, 2012

(Show your work to receive partial credit)

1) For the following code snippet list the data dependencies and rewrite the code to resolve name dependencies. (15 points)

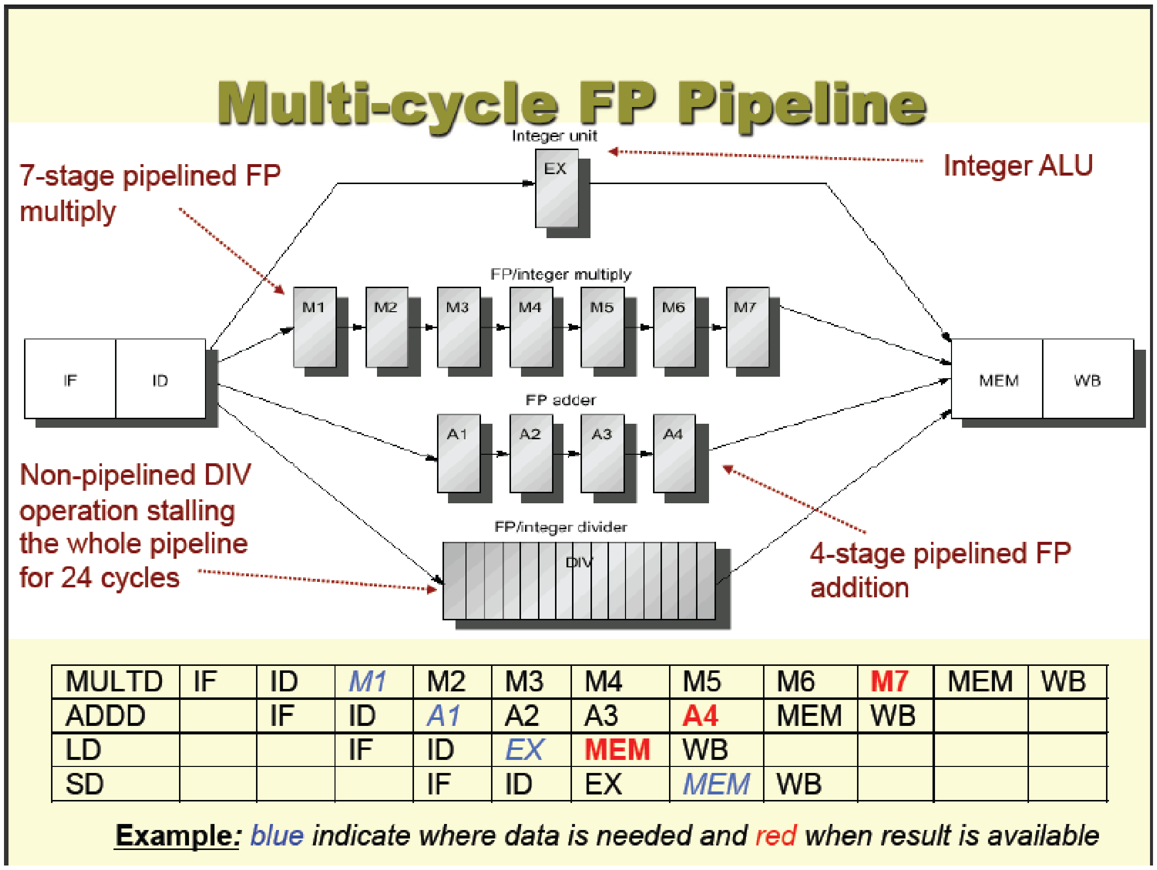
```
Loop:  LD     R2, 0(R7)      ----- 0
       Add    R1, R2, R3     ----- 1
       Sub    R4, R6, R1     ----- 2
       Add    R2, R5, R6     ----- 3
       LD     R4, 32(R1)     ----- 4
       SD     36(R1), R2     ----- 5
       BEQ    R4, Loop       ----- 6
```

Data Dependency –	1 & 2, 1 & 4 and 1 & 5 on R1
	0 & 1, 3 & 5 on R2
	4 & 6 on R4
Name Dependencies-	Anti-dependence between 1 and 3 because of R2
	Output dependence between 2 and 4 because of R4
	Output dependence between 0 and 3 because of R2

Name dependencies can be resolved by renaming registers R2 and R4 with temporary registers T2 and T4

```
Loop:  LD     R2, 0(R7)      ----- 0
       Add    R1, R2, R3     ----- 1
       Sub    R4, R6, R1     ----- 2
       Add    T2, R5, R6     ----- 3
       LD     T4, 32(R1)     ----- 4
       SD     36(R1), T2     ----- 5
       BEQ    T4, Loop       ----- 6
```

2) From the course slides. (Multi-cycle FP pipeline for MIPS) (45 points)



Multi-cycle FP Pipeline

Integer unit
EX
Integer ALU
7-stage pipelined FP multiply
FP/integer multiply
M1
M2
M3
M4
M5
M6
M7
IF
ID
MEM
WB
FP adder
A1
A2
A3
A4
Non-pipelined DIV operation stalling the whole pipeline for 24 cycles
FP/integer divider
DIV
4-stage pipelined FP addition

MULTD	IF	ID	*M1*	M2	M3	M4	M5	M6	**M7**	MEM	WB
ADDD		IF	ID	*A1*	A2	A3	**A4**	MEM	WB		
LD			IF	ID	*EX*	**MEM**	WB				
SD				IF	ID	EX	*MEM*	WB			

Example: *blue indicate where data is needed and red when result is available*

- Use the instruction latencies as indicated in the selected slide to first show all the stalls that is present in the following piece of code if the branch is not taken.
- Now unroll this loop as many times as needed and schedule the instructions to remove all the stalls. You may rename registers i.e. use new registers and/or change the immediate/offset values. You can ignore structural hazards as well.
- What is the speed up that you achieved after unrolling and scheduling?

```
Loop:     LD        F0, 0(r0)
          LD        F2, 0(r2)
          MULTD     F4, F0, F2
          LD        F6, 0(r4)
          ADDD      F8, F4, F6
          SD        0(r6), F8
          ADDI      r0, r0, #4
          ADDI      r2, r2, #4
          ADDI      r4, r4, #4
          ADDI      r6, r6, #4
          SUBI      r8, r8, #1
          BNEZ      r8, Loop
```

```
                                             Clock
Loop:     LD        F0, 0(r0)                1
          LD        F2, 0(r2)                2
          Stall
          MULTD     F4, F0, F2               4
          LD        F6, 0(r4)                5
          Stall
          Stall
          Stall
          Stall
          Stall
          ADDD      F8, F4, F6               11
          Stall
          Stall
          Stall
          SD        0(r6), F8                15
          ADDI      r0, r0, #4               16
          ADDI      r2, r2, #4               17
          ADDI      r4, r4, #4               18
          ADDI      r6, r6, #4               19
          SUBI      r8, r8, #1               20
          Stall
          BNEZ      r8, Loop                 22
          Stall                              23
```

Unrolling it twice is sufficient to remove all the stalls.

```
Loop:     LD        F0, 0(r0)                1
          LD        F2, 0(r2)                2
          LD        F10, 4(r0)               3
          LD        F12, 4(r2)               4
          MULTD     F4, F0, F2               5
          MULTD     F14, F10, F12            6
          LD        F6, 0(r4)                7
          LD        F16, 4(r4)               8
          ADDI      r0, r0, #8               9
          ADDI      r2, r2, #8               10
          ADDI      r4, r4, #8               11
          ADDD      F8, F4, F6               12
          ADDD      F18, F14, F16            13
```

ADDI	r6, r6, #8	14
SUBI	r8, r8, #2	15
SD	-8(r6), F8	16
BNEZ	r8, Loop	17
SD	-4(r6), F18	18

Unrolled loop is (23*2)/18 = 2.55 times faster than without unrolling with respect to clock cycles consumed.
(Ignoring pipeline fill time)
CPI for a) is 23/12 ≈ 2
CPI for b) is 1
After unrolling the CPI is now the same as the ideal CPI.

3) Tomasulo's Algorithm

Consider the following specifications. (40 points)

FU type	cycles in EX	Number of FUs
Integer	1	1
FP adder	5	1
FP multiplier	8	1
FP Divide	24	1

Assume the following.

- Functional units are not pipelined.
- All stages except EX take one cycle to complete.
- No limit on reservation stations.
- There is no forwarding between functional units. Both integer and floating point results are communicated through the CDB.
- Memory accesses use the integer functional unit to perform effective address calculation.
- All loads and stores will access memory during the EX stage. Pipeline stage EX does both the effective address calculation and memory access for loads/stores.
- There are unlimited load/store buffers and an infinite instruction queue.
- Loads and stores take one cycle to execute. Loads and stores share a memory access unit.
- If an instruction is in the WR stage in cycle x, then an instruction that is waiting on the same functional unit (due to a structural hazard) can begin execution in cycle x , unless it needs to read the CDB, in which case it can only start executing on cycle x + 1.

- Only one instruction can write to the CDB in a clock cycle. Branches and stores do not need the CDB since they don't have WR stage.
- Whenever there is a conflict for a functional unit or the CDB, assume program order.
- When an instruction is done executing in its functional unit and is waiting for the CDB, it is still occupying the functional unit and its reservation station. (meaning no other instruction may enter).
- Treat the BNEZ instruction as an Integer instruction. Assume LD instruction after the BNEZ can be issued the cycle after BNEZ instruction is issued due to branch prediction

Fill in the execution profile for the code given in the table which includes the cycles that each instruction occupies in the IS, EX, and WR stages and comments to justify your answer such as type of hazards and the registers involved.

#	Instruction	IS	EX	WR	Comments
1	LD F0, 0(r0)	1	2	3	
2	ADDD F2, F0, F4	2	4-8	9	RAW on F0 from #1
3	MULTD F4, F2, F6	3	17-24	25	RAW on F4 from #2 Structural Hazard from MULT at #7 (Only one functional Unit)
4	ADDD F6, F8, F10	4	9-13	14	Structural Hazard from Adder Add instruction #2
5	DADDI r0,r0, #8	5	6	7	
6	LD F1, 0(r1)	6	7	8	
7	MULTD F1, F1, F8	7	9-16	17	RAW on F1 from #6
8	ADDD F6, F3, F5	8	14-18	19	Structural Hazard from adder Add instruction #4
9	DADDI r1, r1, #8	9	10	11	