HIGH PERFORMANCE COMPUTER ARCHITECTURE exam 07-07-2009 MATRIC.NO.________________

(REVISED 24/10/2014) SURNAME__________________

NAME__________________

1) (7/40) In the following fragment of code, let's identify: i) true dependencies; ii) anti-dependencies; iii) output dependencies; iv) use renaming technique to prevent the problems that are caused by dependencies.

```
LW      R1, 0(R7)
ADD     R1, R2, R4
SUBI    R2, R4, 25
ADD     R4, R1, R3
ADDI    R1, R1, 30
```

2) (16/40) In the following sequence of execution we have in-order dispatch and in-order complete for a two-way superscalar processor; answer the following questions:
i) identify the most probable reason why the instruction I2 cannot enter into Issue+Execute+Writeback before cycle 4;
ii) would out-of-order complete and/or dispatch permit to avoid the delay of I2 in the decode stage (explain why yes or why not)
iii) identify the most probable reason why the instruction I6 cannot enter into the commit stage before cycle 9;
iv) would the out-of-order complete and/or dispatch permit to avoid the dispatch-delay or complete-delay of I6 (explain why yes or why not)

Decode			**Issue+Execute+Writeback**				**Commit**		**Cycle**
Slot1	Slot2	**(Dispatch)**	FU1	FU2	FU3	**(Complete)**	Slot1	Slot2	
I1	I2								1
	I2				I1				2
	I2				I1				3
I3	I4			I2					4
I5	I6			I4	I3		I1	I2	5
					I3				6
			I5	I6			I3	I4	7
			I5						8
							I5	I6	9

3) (17/40) Given the sequence P1: R, P2: R, P1: W, P2: W, P1: W, P2: W (Px:R = read by the processor Px, Px:W write by the processor Px), with respect to a certain variable 'a ', show for each processor the sequence of states, and transactions on the bus that occur in a multiprocessor UMA with write-back cache of each processor and in the case of the coherence protocol Dragon and in the case of the coherence protocol MESI. When the transactions costs are Cbusrd=Cbusrdx=150, Cbusupg=40, Cbusupd=20, Cflush=20, what is the total cost in case of Dragon and in case of MESI ?

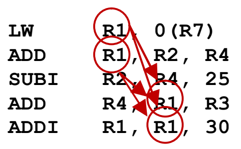
HIGH PERFORMANCE COMPUTER ARCHITECTURE exam 07-07-2009

SOLUTION (REVISED 24/10/2014)

EXERCIZE 1

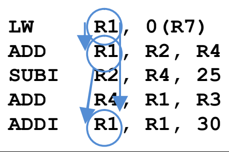
First of all, let's recall that dependencies become hazards (e.g., RAW, WAR, WAW) only when the code is executed on real hardware. In this exercise, we only identify dependencies (i.e., in a very general way).
Please note that the code (i.e., the instructions) could be later parallelized in any permitted way.

i) TRUE DEPENDENCIES



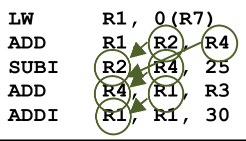
```
LW      R1, 0(R7)
ADD     R1, R2, R4
SUBI    R2, R4, 25
ADD     R4, R1, R3
ADDI    R1, R1, 30
```

ii) OUTPUT DEPENDENCIES



```
LW      R1, 0(R7)
ADD     R1, R2, R4
SUBI    R2, R4, 25
ADD     R4, R1, R3
ADDI    R1, R1, 30
```

iii) ANTI-DEPENDENCIES



```
LW      R1, 0(R7)
ADD     R1, R2, R4
SUBI    R2, R4, 25
ADD     R4, R1, R3
ADDI    R1, R1, 30
```

iv) RENAMED CODE

A possible solution is (Px indicates a renamed register)

```
LW      P2, 0(P1)
ADD     P5, P3, P4
SUBI    P6, P4, 25
ADD     P8, P5, P7
ADDI    P9, P5, 30
```

EXERCIZE 2

i) Since I2 and I1 are on two different columns in the execution stage, i.e. they use different functional units, it is unlikely that there is a resource conflict. It is instead quite likely that there is a TRUE DEPENDENCY on the data, i.e. I1's result is necessary to execute I2.

ii) It is not possible to avoid I2's delay since true dependency cannot be resolved with out-of-order dispatch nor with out-of-order complete: the data coming from the "producer instruction" must be received by a "consumer" instruction: the out-of-order dispatch and out-of-order complete can only help when dealing with INDEPENDENT instructions.

iii) By definition, out-of-order complete implies that I6 should wait that I5 terminates its execution before I6 enters the "Commit" stage. Since both I5 and I6 had entered the pipeline at the same cycle, they must also exit the pipeline in the same order.

iv) From the analysis of the given execution, we can make the following deductions:

- I2 wait in the IW (Instruction Window) until I1 terminates its execution to satisfy a dependency I1 → I2;
- I3 exits IW as soon as I1 has freed up FU3;
- I4 exits IW as soon as I2 has freed up FU2
- I5 does not need to stay in IW and can therefore enter immediately into Issue+Execute+Writeback
- I6 must wait the I4 frees up FU2 before entering into Issue+Execute+Writeback

When both out-of-order dispatch and execute are used the modified diagram is the following:

Decode		**Instr.Window**	**Issue+Execute+Writeback**			**ROB Completed**	**Commit**		**Cycle**
Slot1	Slot2		FU1	FU2	FU3		Slot1	Slot2	
I1	I2								1
I3	I4	I1,I2			I1				2
I5	I6	I2,I3,I4			I1	I1			3
		I4,I5,I6	I5	I2	I3	I2	I1		4
		I6	I5	I4	I3	I3,I4,I5	I2		5
				I6		I6	I3	I4	6
							I5	I6	7

Therefore, the I6 delay in the Issue+Execute+Writeback stage can be reduced by anticipating previous instructions (I3 and I5 in particular). The I6 delay in the Commit stage can be reduced since now I6 must only wait for I4. Please note that out-of-order complete allows I3+I4+I5 to terminate their execution simultaneously, however the superscalar processor is only two-ways (we have only to "columns" for the write-back) so I5 must wait one extra cycle.

EXERCIZE 3

Dragon Protocol

CPU-Action	P1	P2	Bus-Action	Cost
P1:R	E	-	BusRd(S')	150
P2:R	Sc	Sc	BusRd(S)	150
P1:W	Sm	Sc	BusUpd(S)	20
P2:W	Sc	Sm	BusUpd(S)	20
P1:W	Sm	Sc	BusUpd(S)	20
P2:W	Sc	Sm	BusUpd(S)	20

TOTAL **380**

MESI Protocol

CPU-Action	P1	P2	Bus-Action	Cost
P1:R	E	I	BusRd(S')	150
P2:R	S	S	BusRd(S)/Flush	150+20
P1:W	M	I	BusUpgr(S)	40
P2:W	I	M	BusRdX(S)/Flush	150+20
P1:W	M	I	BusRdX(S)/Flush	150+20
P2:W	I	M	BusRdX(S)/Flush	150+20

TOTAL **870**