Q1. [8 marks] True or false

Determine whether the following statements are true or false. **Circle** the answer. NO further explanation for the answer is needed.

i. A 100MIPS computer is NOT necessarily faster than an 95MIPS computer when they are running the same Java program (MIPS: Million Instructions Per Second)

[✓ True] False

ii. Instruction count in a program depends on the instruction set architecture used.

[✓ True] False

iii. To run on a dynamic scheduling processor, such as an IBM 360/91 system which implements the Tomasulo algorithm, an ISA must have 16 or more registers.

True [✓ False]

iv. If we use an instruction in the branch target path to fill a delay slot, we may need to duplicate the instruction.

[✓ True] False

Q2. [24 marks] Multiple choices

Choose ONE answer and write the corresponding letter in the blank space at the beginning of the question.

i. _A__ Which one of the following types of hazards CANNOT be removed by register renaming?

A. RAW hazards
B. WAR hazards
C. WAW hazards
D. None of the above
E. All of the above

ii. _E__ A 2048-entry (5, 3) correlating predictor would use

A. 5001 bits
B. 8192 bits
C. Between 16000 to 17000 bits
D. Between 30000 to 40000 bits
E. More than 100K bits

iii. _A__ If we run the following instruction sequence on a MIPS processor with the 5-stage pipeline WITH forwarding, how many stalls (bubbles) will this code sequence incur?

```
LW     r1, 100(r2)
LW     r3, 200(r2)
ADD    r5, r1, r3
SW     r5, 300(r2)
```

A. 1
B. 2
C. 3
D. 4
E. 5

iv. _C__ Which one of the following is NOT a hazard on **most of the modern computers**? (Hint: not just the MIPS computers)

A. WAW
B. RAW
C. RAR
D. WAR

v. _B__ There are several classes of ISAs –accumulator based ISAs, stack based ISAs, and General Purpose Register based ISAs. The MIPS computers (using MIPS instruction set) belong to

A. Accumulator based ISA
B. General Purpose Register (GPR) based ISA
C. Stack based ISA

vi. _B__ Which one of the following items does NOT DIRECTLY affect a processor's ability to exploit ILP? Assume that the computer system has only one single-core processor, only one single-threaded program runs on the system, and the physical memory is large enough to hold all the active pages thus no virtual memory swapping is needed.

A. The design of the branch predictor
B. The clock cycle time
C. The number of instructions a processor can issue every cycle
D. The optimization level used for compiling the programs

Q3 [10 marks] Dependence among instructions

Identify all dependences by type (data dependence, name dependence, control dependence) in the following code segment; list the two instructions involved; identify which instruction is dependent; and, if there is one, name the storage location involved.

```
1    LW     R1, 45(R2)
2    ADD    R7, R2, R1
3    SUB    R8, R1, R6
4    OR     R1, R5, R8
5    BNEZ   R7, branch_target
6    XOR    R3, R3, R4
```

Please use the following table to answer. You may not need all the rows.

	Dependence type	Independent instruction	Dependent instruction	Storage location
1 (example)	Data	1(LW)	2(ADD)	R1
2	Data	1	3	R1
3	Name (output)	1	4	R1
4	Data	2	5	R7
5	Name (anti)	2	4	R1
6	Data	3	4	R8
7	Name (anti)	3	4	R1
8	Control	5	6	
9				
10				

Instruction examples: In some questions, you may need to read or write MIPS instructions in assembly language. Below are some examples of instructions in assembly:

Load a word from memory address REGS[R2]+100 to R1:	`LW R1, 100(R2)`
Load 64 bits from memory to F0 (double precision):	`LD F0,100(R2)`
Store the word in R3 to memory:	`SW R3, 100(R4)`
Add R6 and R7 and assign the result to R5:	`ADD R5, R6, R7`
Add R6 and an immediate, and assign the result to R5:	`ADD R5, R6, 200`
Add F1 and F2 and assign the result to F3 (single-precision):	`ADD.S F3, F1, F2`
Branch to address indicated by Label9 if R8 is not zero:	`BNEZ R8, Label9`

You can use ';' to start comments in a line, e.g.,

```
    LW R1, 100(R2); Here is the comment within one line
```

Q4 [9 marks] Pipeline stages

Fill in the pipeline timing chart for the code segment. Assume the pipeline is a 5-stage MIPS pipeline with forwarding from the EX/MEM latch to the input of EX-stage and MEM-stage hardware, and forwarding from the MEM/WB latch to the input of EX-stage and MEM-stage hardware. Use F, D, E, M, W and S for fetch (IF), decode (ID), execute (EX), memory (MEM), write back (WB) and stalls, respectively.

Instruction	Clock Cycles														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LW R2, (R1) (example)	F	D	E	M	W										
ADD R1, R2, R3		F	D	S	E	M	W								
ADD R4, R1, R5			F	S	D	E	M	W							
LW R4, 100(R4)					F	D	E	M	W						
SW R4, 8(R1)						F	D	E	M	W					

Q5 [14 marks] Code scheduling

The following code calculates the floating-point expression E = A * B + C * D, where the memory addresses of A, B, C, D, and E are stored in R1, R2, R3, R4, and R5, respectively:

```
L.S     F0, 0(R1)
L.S     F1, 0(R2)
MUL.S   F0, F0, F1
L.S     F2, 0(R3)
L.S     F3, 0(R4)
MUL.S   F2, F2, F3
ADD.S   F0, F0, F2
S.S     F0, 0(R5)
```

X.S means performing operation X on single-precision data. For example, L.S means loading a single-precision number.

i. [6 marks] Suppose we execute the above code segment on an in-order pipelined machine. **Show the stalls** we need to insert between instructions, and **calculate the number of cycles** this code sequence would take to execute (i.e., the number of cycles between the issue of the first load instruction and the issue of the final store, inclusive). For simplicity, you can think of the "issue" cycle (or the cycle in which an instruction is issued) as the cycle immediately before an instruction enters its execution functional unit for execution.

Consider the memory load/store unit as a functional unit in parallel with the FP ALU units. Assume the following latencies:

Load → FP ALU:	2 cycles
FP multiplication → FP ALU:	4 cycles

FP multiplication → Store:	4 cycles
FP addition → FP ALU:	2cycles
FP addition → Store:	2cycles

As an example, if a load instruction L.S F20, 0(R1) is issued in cycle 1, an ALU instruction ADD.S F20, F20, F20 can be issued in cycle 4 and proceed in the pipeline without stalls. Assume that all functional units are fully pipelined and ignore any write back conflicts on the register file.

```
1       L.S   F0, 0(R1)
2       L.S   F1, 0(R2)
3       Stall       ; Load → FP ALU:           2 cycles
4         Stall
5       MUL.S   F0, F0, F1
6       L.S   F2, 0(R3)
7       L.S   F3, 0(R4)
8       Stall       ; Load → FP ALU:           2 cycles
9       Stall
10      MUL.S   F2, F2, F3
11      Stall       ; FP multiplication → FP ALU:   4 cycles
12      Stall
13      Stall
14      Stall
15      ADD.S   F0, F0, F2
16      Stall       ; FP addition → Store:         2cycles
17      Stall
18      S.S   F0, 0(R5)
```

Totally, 18 cycles.

ii. [8 marks] Reorder the instructions in the code sequence to minimize the execution time. Show the new instruction sequence, and indicate the position and number of stalls we need to insert between instructions.

```
1       L.S   F0, 0(R1)
2       L.S   F1, 0(R2)
3       L.S   F2, 0(R3)
4       L.S   F3, 0(R4)
5       MUL.S   F0, F0, F1
6       Stall
7       MUL.S   F2, F2, F3
8       Stall       ; FP multiplication → FP ALU:   4 cycles
9       Stall
10      Stall
11      Stall
12      ADD.S   F0, F0, F2
```

```
13      Stall       ;FP addition → Store:        2cycles
14      Stall
15      S.S   F0, 0(R5)
Totally 15 cycles.
```

Q6. [11 marks] Instruction set design

A given processor has 56 instructions (corresponding to 56 operation codes) and 128 registers, and uses 16-bit immediates in its ISA. In the instruction set, there are four types of instructions as listed below:

Type A: takes one source register, uses one destination register,
Type B: takes two source registers, uses one destination register,
Type C: takes one source register, one immediate, uses one destination register,
Type D: takes one immediate, uses one destination register.

Assume that the ISA requires that all instructions be a multiple of 8 bits (1 byte) in length, and the operation codes (opcodes) are of fixed length (i.e., the ISA does not use shorter opcodes for some instructions and longer opcodes for others).

i. (3 marks) How many bits do we need to use to encode the opcodes?
6

ii. (4 marks) How many **bytes** do we need to use to encode the Type-B instruction?
$6+3*7 = 27 \rightarrow$ 4 bytes

iii. (4 marks) How many **bytes** do we need to use to encode the Type-D instruction?
$6+7+16 = 29 \rightarrow$ 4 bytes

Q7. [16 marks] Scoreboarding

In a scoreboarding system, instructions go through 4 steps in the pipeline—issue (ID1), read operands (ID2), execution (EX) and write result (WB).

i. (4 marks) In which step does the scoreboard check for WAR hazards?
Write result (WB)

ii. (4 marks) In which step does the scoreboard check for structural hazards?
Issue (ID1)

iii. (10 marks) The scoreboard checks for the WAW hazard in the issue step. Can it postpone the checking for WAW hazard to the write result step? Why?
No.
The traditional scoreboarding design checks the WAR hazard in the Write Result (WB) step. If no two issued instructions have a WAW hazard, it is rather simple to check for WAR hazards. Suppose an instruction i enters the WB stage and is about to write a result to register r. If there is another instruction j in the pipeline that reads r and the scoreboard shows that j's source operand corresponding to r is ready, there is a WAR

hazard. Otherwise, there is no WAR hazard. This reasoning relies on the assumption that no two instructions writing to the same register are allowed to go through the issue stage and become active in the pipeline. With this assumption, if instruction j reads r but r is not ready, j should see r's value as written by i, because i is the only active instruction that writes r, or j has already read its operands. In order to make this assumption valid, WAW must be checked for and prevented in the issue step.

Q8. [8 marks] General performance

Before and after an architecture "improvement", a computer's CPU 2006 benchmark scores on three programs, bzip2, gcc, and astar, are as follows. The scores are the ratio of performance to the reference machine Sun SPARC Enterprise M8000.

Benchmark	**Score before improvement**	**Score after improvement**
401.bzip2	39	50
403.gcc	36	40
473.astar	19	16

i. (6 marks) Using geometric mean of the scores on the above three programs to summarize the performance, compute the speedup after the improvement. Write down the equations and intermediate steps when you develop the result. (Hint: SPEC benchmark scores are already ratios against a reference machine, and they specify the performance, not execution time, of a computer. A higher score means a faster computer.)

$$\text{GM before improvement} = \sqrt[3]{39 \times 36 \times 19} = 29.88$$

$$\text{GM after improvement} = \sqrt[3]{50 \times 40 \times 16} = 31.75$$

$$\text{Speedup} = \frac{31.75}{29.88} = 1.06$$

ii. (2 points) Based on the speedup we get in the last question (Q8.i), is the computer faster after the improvement?

Yes.