0907731 Advanced Computer Architecture (Spring 2018)
Final Exam

الاسم: ..... رقم التسجيل: ..... رقم التسلسل: .....

==========

Instructions: Time **80** min. Open book and notes exam. No electronics. Please answer all problems in the space provided and limit your answer to the space provided. No questions are allowed. There are eight problems and each problem is for 5 marks.

==========

P1. Assume that we make an enhancement to a computer that improves some mode of execution by a factor of 10. Enhanced mode is used 50% of the time, measured as a percentage of the execution time *when the enhanced mode is in use*. Recall that Amdahl's law depends on the fraction of the original, *unenhanced* execution time that could make use of enhanced mode. Thus, we cannot directly use this 50% measurement to compute the speedup using Amdahl's law. What is the speedup we have obtained from the enhancement?

For the enhanced run: s = 10

(1-f) = f/s => 10 – 10f = f => 11f = 10 => f = 10/11

Speedup = 1 / (1-f + f/s) = 1 / (1/11 + 10/11/10) = 1 / (2/11) = 11/2 = 5.5

P2. Assume that the following code sequence is executed by a dual-issue speculative pipelined processor. This processor uses reservation stations, common data bus, and reorder buffer. The fetch stage takes one cycle and the issue stage takes one cycle. The integer latency is 1 cycle and the memory latency is 2 cycles (1 cycle for address calculation and 1 cycle for data memory access). The processor has one address calculation unit, one memory access unit, and one integer ALU units. Using the multi-cycle pipeline diagram below, specify the execution of these instructions in this processor pipeline.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
`lw R2, 0(R1)`	**F**	**I**	**A**	**M**	**W**	**C**									
`lw R3, 0(R2)`	**F**	**I**		**A**	**M**	**W**	**C**								
`add R4, R2, R3`		**F**	**I**				**E**	**W**	**C**						
`sw R4, 0(R2)`		**F**	**I**		**A**				**C**						

P3. For the purpose of this problem, we assume that we have 512-byte cache with 64-byte blocks. We also assume that the main memory is 2 KB large. We can regard the memory as an array of 64-byte blocks: M0, M1, …, M31. The following table shows the memory blocks that can reside in different cache blocks if the cache was fully associative. Show the contents of the table if the cache is organized as direct-mapped cache.

Fully-associative cache			
Cache block	**Set**	**Way**	**Memory blocks that can reside in cache block**
0	0	0	M0, M1, …, M31
1	0	1	M0, M1, …, M31
2	0	2	M0, M1, …, M31
3	0	3	M0, M1, …, M31
4	0	4	M0, M1, …, M31
5	0	5	M0, M1, …, M31
6	0	6	M0, M1, …, M31
7	0	7	M0, M1, …, M31

Direct-mapped cache			
Cache block	**Set**	**Way**	**Memory blocks that can reside in cache block**
0	**0**	**0**	**M0, M8, M16, M24**
1	**1**	**0**	**M1, M9, M17, M25**
2	**2**	**0**	
3	**3**	**0**	**…**
4	**4**	**0**	
5	**5**	**0**	
6	**6**	**0**	
7	**7**	**0**	**M7, M15, M23, M31**

P4. A traditional VLIW processor accepts long instructions that have the following five fields:

Branch	ALU	ALU	Memory	Memory

Show the best schedule for the following 10 operations into such 5-operation instructions. Assume that the processor has full forwarding paths, resolves the branch instructions in the execute stage, executes branch and ALU instructions in one cycle, and executes memory instructions in two cycles (address calculation and memory access). Show your schedule by writing the instruction numbers (`I1` through `I10`) in the table to the right of the 10 operations.

```
I1  Loop:   lw    r1, 0(r2)
I2          lw    r3, 1000(r2)
I3          lw    r4, -4(r2)
I4          add   r1, r1, r3
I5          lw    r5, 996(r2)
I6          sw    r1, 2000(r2)
I7          add   r4, r4, r5
I8          addi  r2, r2, -8
I9          sw    r4, 2004(r2)
I10         bne   r2, zero, Loop
```

Branch	ALU	ALU	Memory	Memory
			I1	**I2**
			I3	**I5**
	I4			
	I7	**I8**	**I6**	
I10			**I9**	

P5. Convert the following C-language double-precision code into Vector MIPS. Assume that `x` and `y` are double-precision vectors and their starting addresses are in registers `Rx` and `Ry`, respectively.

```
for (i=0; i<64; i=i+1)
  x[i] = x[i] * y[i] + 2.0;
```

The solution is:

```
.data
fp1:  .double 2.0

.text
      lv      $v1, (Rx)       ;load vector x
      lv      $v2, (Ry)       ;load vector y
      mulv.d  $v3,$v1,$v2     ;vector add
      l.d     $f0, fp1
      addvs.d $v3, $v3, $f0   ;add vector to scalar
      sv      $v3, (Rx)       ;store vector x
```

P6. Given same hardware resources, why does simultaneous multithreading (SMT) processors achieve higher instruction throughput compared to superscalar processors.

SMT processors fetch and execute instructions from multiple threads simultaneously. While superscalar processors, they fetch instructions from a single thread. Therefore, SMT processors can find more independent instructions to execute concurrently. Thus, they achieve higher instruction throughput.

P7. For the write-back snoopy cache coherence protocol described in the class, complete the following table for Processors P1 and P2 connected through a snoopy bus to the shared memory. Assume that Addresses A1 and A2 map to same cache block. Enter (M, S, or I) in the "State" field, (A1 or A2) in the "Addr" field, (Read miss, Write miss, or Write back) in the "Action" field, and the processor number in the "Proc" field.

	P1			P2			Bus				Memory	
Step	State	Addr	Value	State	Addr	Value	Action	Proc	Addr	Value	Addr	Value
P2 writes 7 to A1				**E**	**A1**	**7**	**Write miss**	**P2**	**A1**			
P1 reads A1	**S**	**A1**	**7**	**S**	**A1**	**7**	**Read miss** **Write back**	**P1** **P2**	**A1** **A1**	**7**	**A1**	**7**
P2 writes 9 to A2	**I**	**A1**		**E**	**A2**	**9**	**Write miss**	**P2**	**A2**			
P1 writes 13 to A1	**E**	**A1**	**13**	**I**	**A2**		**Write miss** **Write back**	**P1** **P2**	**A1** **A2**	**9**	**A2**	**9**

P8. Assume a hypothetical GPU with the following characteristics:

- Clock rate 2.0 GHz
- Contains 16 SIMD processors, each containing 16 single-precision floating-point units
- Has 400 GB/sec off-chip memory bandwidth

A) Without considering memory bandwidth, what is the peak single-precision floating-point throughput for this GPU in GFLOPS/s, assuming that all memory latencies can be hidden?

Peak performance = 16 x 16 x 2.0 = 512 GFLOPS/s

B) Is this throughput sustainable given the memory bandwidth limitation? Assume that each single precision operation requires two operands and outputs one result.

Required memory bandwidth = (2+1) x 4 x 512 = 6.1 TB/s

The required memory bandwidth is much larger the off-chip memory bandwidth. Therefore, this peak performance is not sustainable.

<Good Luck>