German University in Cairo
Faculty of Media Engineering and Technology
Dr. Cherif Salama

Bar Code

CSEN 601 Computer Architecture

Final Exam Solution

Instructions: Please Read Carefully Before Proceeding.

1. The allowed time for this exam is **3 hours** (180 minutes).
2. This exam includes a **5% bonus**
3. Non-programmable calculators are allowed.
4. No books or other aids are permitted for this test.
5. This exam booklet contains 11 pages, including this one. An appendix sheet and an extra sheet of scratch paper are attached and have to be kept attached. **Note that if one or more pages are missing, you will lose their points. Thus, you must check that your exam booklet is complete**.
6. Please write your solutions in the space provided. If you need more space, **please use the back of the sheet containing the problem or the extra sheet and make an arrow indicating that.**
7. When you are told that time is up, please stop working on the test.

All the best.

Please, do not write anything on this page.

Question	1	2	3	4	Total
Maximum Marks	20	30	40	15	100
Earned Marks	20	30	40	15	105

Question 1 **(20 marks)**

[20 marks] Mark each of the following statements as true (✓) or false (✗). Unless otherwise stated, assume that all statements are in the context of the MIPS architecture.

#	Statement	Answer
1	The `beq, bne, J, and JAL` instructions use PC-relative addressing to specify the target address	✗
2	The only difference between the `J` and the `JAL` instructions is that the latter automatically saves the return address to the stack	✗
3	In the single cycle implementation, the `SW` instruction is the longest instruction and as such it is the one responsible for determining the clock period	✗
4	In the pipelined implementation, all instructions including R-type instructions have to go through all 5 pipeline stages including the MEM stage	✓
5	For LW and SW instructions, memory address computation is done in the ID stage, while actual memory access is done in the MEM stage	✗
6	Forwarding can completely eliminate stalls between R-type instructions	✓
7	Among other responsibilities, a hazard detection unit is responsible for forwarding the appropriate operands when a data dependency is detected	✗
8	In the studied pipelined implementation, structural hazards never show up	✓
9	One way to reduce the penalty associated with control hazards, is to compute the branch outcome as early as possible; namely during the IF stage	✗
10	The branch target buffer is like a cache for previously computed branch target addresses. It is used in conjunction with branch prediction to avoid branch hazard penalties	✓
11	To simplify exception handling, the MIPS pipeline is designed such that exceptions can only occur in the EX stage	✗
12	A 4-way superscalar processor has a peak IPC of 4	✓
13	Superscalar processors are processors having wider pipelines	✓
14	Dynamic (out-of-order) superscalar processors guarantee that instructions are written back in strict program order	✓
15	Loop unrolling is a compiler optimization capable of exposing more parallelism and reducing control overhead at the expense of using more memory and registers	✓
16	Flash memory is faster than magnetic storage due to better throughput	✗
17	DMA stands for dynamic memory access	✗
18	RAID stands for Rapid Array of Independent Disks	✗
19	RAID systems are used to improve availability, without affecting performance	✗
20	RAID level 0 does not offer any redundancy at all	✓

Question 2 **(10+10+10=30 marks)**

Consider the following MIPS code where $s0 contains the value 500, which is the starting address of a 12-elements array containing the values from 1 to 12.

```
    addi  $t0, $zero, 536
    addi  $t1, $zero, 7
L1: add   $t2, $s0, $zero
L2: lw    $t3, 0($t2)
    addi  $t3, $t3, 1
    sw    $t3, 0($t2)
    addi  $t2, $t2, 4
    bne   $t0, $t2, L2
    addi  $t1, $t1, -1
    bne   $t1, $zero, L1
```

2.1 [10 marks] Answer the following questions:

a. What are the contents of $t0, $t1, $t2, $t3, and $s0 at the end of the program above?
b. What are the contents of the array starting at address 500 at the end of the program?
c. How many instructions were executed in total?
d. The code above uses pointers to access the array. Re-write it to use indices instead.

Solution:

a. $t0 = 536, $t1 = 0, $t2 = 536, $t3 = 16, and $s0 = 500
b. The array contains the values: 8, 9, 10, 11, 12, 13, 14, 15, 16, 10, 11, and 12
c. In total instructions executed = (5 * 9 + 3) * 7 + 2 = 338 instructions
d. Using indices the above code becomes:

```
    addi  $t0, $zero, 9
    addi  $t1, $zero, 7
L1: add   $t2, $zero, $zero
L2: sll   $t2, $t2, 2
    add   $t4, $s0, $t2
    lw    $t3, 0($t4)
    addi  $t3, $t3, 1
    sw    $t3, 0($t4)
    addi  $t2, $t2, 1
    bne   $t0, $t2, L2
    addi  $t1, $t1, -1
    bne   $t1, $zero, L1
```

2.2 [10 marks] Based on the assembly code above, how many times is the bne instruction in line 8 executed? What about the bne in line 10? If a 1-bit branch predictor is used to predict the branch outcome of both conditional branch instructions, what is the misprediction percentage? Assume that both bne instructions are initially predicted as not-taken. What if a 2-bit branch predictor is used? In this case assume that both bne instructions are initially considered to be in the weakly-not-taken state (the weakly-not-taken state is the not-taken state that is directly connected to a taken state).

Solution:

The bne instruction in line 8 is executed 9*7 = 63 times.
The bne instruction in line 10 is executed 7 times.

If 1-bit branch prediction is used,
Line 8 bne is misspredicted = 2*7 = 14 times
Line 10 bne is misspredicted 2 times
Total number of misspredictions = 16 times
Misspredicition percentage = 16/70 = 22.86 %

If 2-bit branch prediction is used,
Line 8 bne is misspredicted 8 times
Line 10 bne is misspredicted 2 times
Total number of misspredictions = 10 times
Misspredicition percentage = 10/70 = 14.29%

2.3 [10 marks] Translate the assembly code above into machine code (binary) knowing that the opcodes for add, addi, lw, sw, and bne are 0, 8, 35, 43, and 5 respectively and that the function code for add is 32. It is enough to translate one instruction for each opcode (e.g., there is no need to translate all 5 addi instructions, one addi instruction is enough as shown in the solution table below)

Solution:

Line#	Instruction	Equivalent Machine Code
1	`addi $t0, $zero, 536`	001000 00000 01000 0000 0010 0001 1000 (I-Format)
3	`L1: add $t2, $s0, $zero`	000000 10000 00000 01010 00000 100000 (R-Format)
4	`L2: lw $t3, 0($t2)`	100011 01010 01011 0000 0000 0000 0000 (I-Format)
6	`sw $t3, 0($t2)`	101011 01010 01011 0000 0000 0000 0000 (I-Format)
8	`bne $t0, $t2, L2`	000101 01000 01010 1111 1111 1111 1011 (I-Format)

Question 3 **(10+7+15+8=40 marks)**

3.1 [10 marks] Write a **recursive** MIPS **procedure** that locates the first occurrence of a character in a null-terminated C string. The function takes two parameters: the address of the string and the character. It should either return the index where the character was found or -1 if the character is not found. Write a minimal test **program** showing how to call this procedure to search for the character "A" (ASCII code 65) in a C string starting at address 1000. Do not use any instructions not listed in the appendix.

Solution:

Recursive MIPS procedure

```
loc:  lbu   $t0, 0($a0)
      beq   $t0, $a1, L1
      beq   $t0, $zero, L2
      addi  $sp, $sp, -4
      sw    $ra, 0($sp)
      addi  $a0, $a0, 1
      jal   loc
      lw    $ra, 0($sp)
      addi  $sp, $sp, 4
      addi  $t1, $zero, -1
      beq   $v0, t1, L3
      addi  $v0, $v0, 1
      jr    $ra
L1:   addi  $v0, $zero, 0
      jr    $ra
L2:   addi  $v0, $zero, -1
L3:   jr    $ra
```

Test program:

```
      addi  $a0, $zero, 1000
      addi  $a1, $zero, 65
      jal   loc
```

3.2 [7 marks] What are the IF.Flush and ID.Flush control signals? Which unit is responsible for generating them? Briefly describe a case where they are used.

Solution:

The IF.Flush is a signal that controls when the IF/ID pipeline register should be cleared. The ID.Flush is a signal that controls when the control fields of ID/EX pipeline register should be cleared. The control unit is responsible for generating both signals. By default both signals are equal to 0. The IF.Flush is set to 1 when the control unit detects that an incorrect branch direction has been taken (due to incorrect prediction). The ID.Flush is set to 1 when the control unit detects an exception that requires the cancellation of the instruction in the ID stage.

3.3 [15 marks] Assume a 5-stage pipelined MIPS implementation and consider the following instruction sequence:

```
ADD   $s1, $s2, $s3
LW    $s1, 4($s1)
LW    $s2, 12($s4)
SUB   $s4, $s1, $s2
```

a. Find all the data dependencies and their types.
b. Assuming the sequence of instructions is executed correctly, that the initial values of $s2, $s3, and $s4 are 100, 200, and 88 respectively and that memory locations 100 and 304 contains the values 710 and 320 respectively, what will be the final values of $s1, $s2, $s3, and $s4?
c. List the hazards assuming there is neither forwarding nor hazard detection units. What we be the final values of $s1, $s2, $s3, and $s4 in this case?
d. Add ***nop*** instructions to eliminate the hazards in the previous case.
e. Assume there is full forwarding. Indicate hazards and add ***nop*** instructions to eliminate them.
f. Assuming a clock period of 100 ps, what is the total time to execute this instruction sequence correctly without forwarding? What is the total time in case of full forwarding? What is the speed-up achieved by adding full forwarding? Assume that the clock period is increased by 10% in case of full forwarding.

Solution:

a. **RAW dependencies**: $s1 between I1 and I2, $s1 between I2 and I4, $s2 between I3 and I4
WAW dependencies: $s1 between I1 and I2
WAR dependencies: $s2 between I1 and I3, $s4 between I3 and I4
b. $s1 = 320, $s2= 710, $s3 = 200, and $s4 = -390
c. All the RAW dependencies above are hazards. $s1 = unknown (it depends on in the initial value of $s1 itself and on the memory contents), $s2= 710, $s3 = 200, and $s4 = 200
d.

```
ADD  $s1, $s2, $s3
nop
nop
LW   $s1, 4($s1)
LW   $s2, 12($s4)
nop
nop
SUB  $s4, $s1, $s2
```

e. In case of full forwarding the only hazard remaining is the $s2 RAW hazard between I3 and I4.

```
ADD  $s1, $s2, $s3
LW   $s1, 4($s1)
LW   $s2, 12($s4)
nop
SUB  $s4, $s1, $s2
```

f. Without forwarding, the total time = 12 * 100 ps = 1200 ps
With full forwarding, the total time = 9 * 110 ps = 990 ps
The speedup achieved by adding full forwarding = 1200 / 990 = 1.11

3.4 [8 marks] For the instruction sequence shown in 3.3:

a. What is the total time to execute the same sequence using a single cycle implementation with a clock period of 500 ps? What is the speedup of pipelining (with full forwarding) with respect to the non-pipelined implementation?
b. Assuming the above sequence is executed a huge number of times, what is the speedup (in terms of IPC) achieved when going from a 1-issue (pipelined with full forwarding) processor to a 3-issue statically scheduled processor **without any restrictions** on the type of instructions that can grouped in each instruction packet. Assume that the compiler is free to rearrange instructions in case of the 3-issue processor. Show the compiler schedule you assumed.

Solution:

a. Without pipelining (single cycle), the total time = 4*500 ps = 2000 ps
The speedup achieved by pipelining with full forwarding = 2000 / 990 = 2.02
b. In case of the 1-issue processor the IPC would be 0.8 (since the above sequence has 4 instructions and requires 5 clock cycles). In case of the 3-issue processor, the compiler could schedule the instructions as follows:

Instruction 1	Instruction 2	Instruction 3
ADD $s1, $s2, $s3	LW $s2, 12($s4)	nop
LW $s1, 4($s1)	nop	nop
nop	nop	nop
SUB $s4, $s1, $s2	nop	nop

In this case the IPC would be: 4/4 = 1. This indicates in a speedup of 1/0.8 = 1.25.

Question 4 **(8+3+4=15 marks)**

4.1 [8 marks] Consider a 200 MB/s magnetic hard drive having 6 double-sided platters (disks). The platters rotate at 10000 rpm. Each platter's surface is divided into 12000 tracks and each track is divided into 100 sectors (blocks). The storage capacity of each sector is 8192 bytes. The actual average seek time is 3ms. Assume the controller overhead is zero and that the disk is initially idle. Answer the following questions:

a. What is the total storage capacity of this hard drive?
b. How much time does it take to read a 8 MB file whose blocks are scattered randomly on the disk?
c. Defragmentation is the process of re-arranging file blocks to occupy consecutive physical blocks instead of being scattered. What is the speed-up achieved when reading the same 8 MB file after defragmentation?

Solution:

a. Total capacity = 6 x 2 x 12000 x 100 x 8192 bytes = 109.86 GBytes
b. Time to read one 8192 block = average actual seek time (3 ms) + average rotational latency (½ / (10000/60) = 3 ms) + transfer time (8192/200 MB/s = 0.04096 ms) = 6.04096 ms.
The 8 MB file has (8 * 1024 * 1024 / 8192) = 1024 blocks.
Therefore the time needed to read the entire file is 6.04096 x 1024 ms = 6.186 seconds.
c. When reading the defragmented 8 MB file, seek and rotation happen only once. The total time in this case would be seek (3 ms) + rotation (3 ms) + transfer (41.943 ms) = 47.943 ms. The speedup is 6186/47.943 = 129.03

4.2 [3 marks] There are 3 types of registers found in I/O controllers. List these register types with a brief description of each.

Solution:

I/O controllers register types:

- **Command** registers: Cause device to do something
- **Status** registers: Indicate what the device is doing and occurrence of errors
- **Data** registers: Transfer data to/from a device

4.3 [4 marks] List and compare both I/O register-mapping mechanisms you studied.

Solution:

There are Two I/O register-mapping mechanisms:

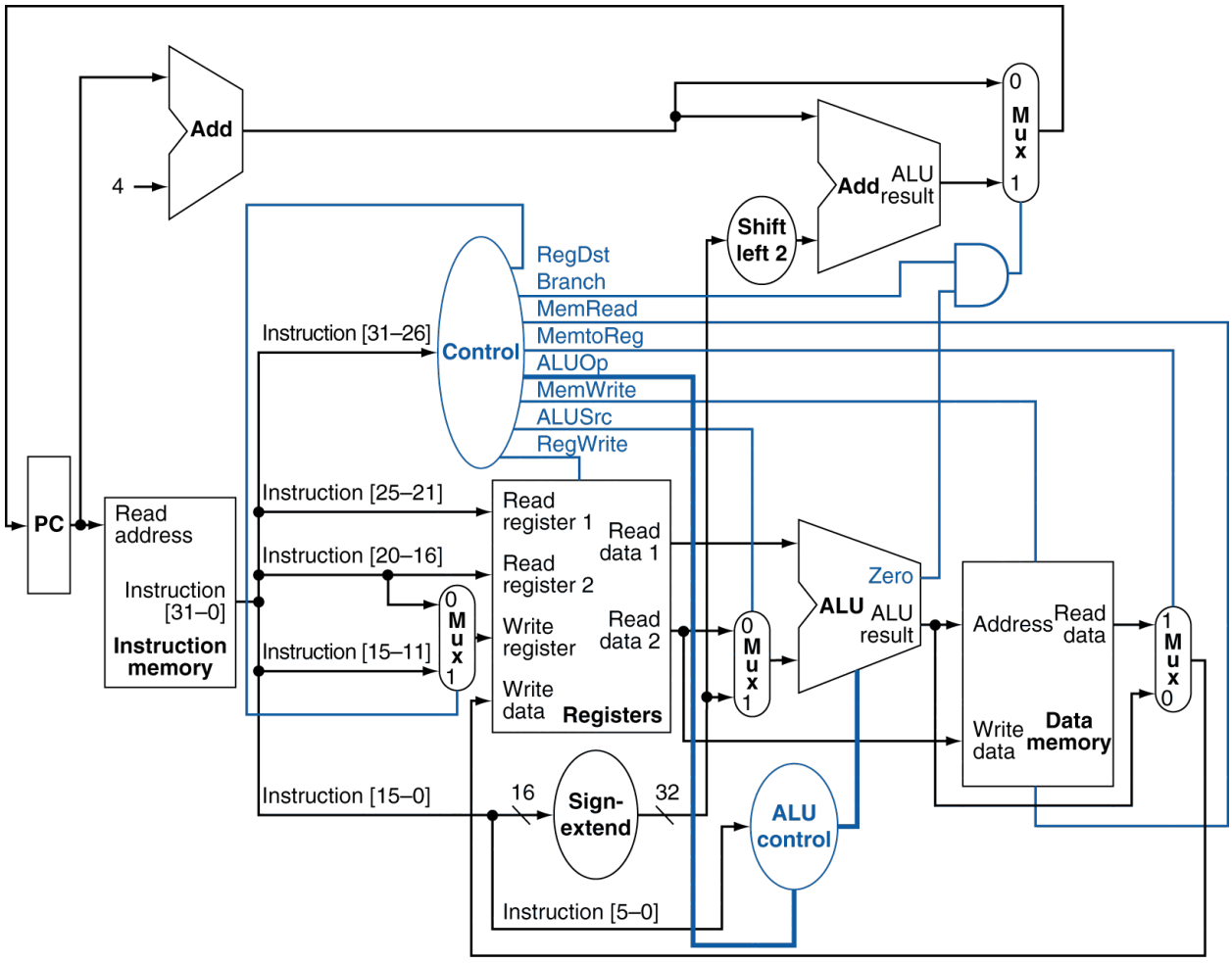
- **Memory-Mapped** I/O
- **Isolated** I/O

	Memory-Mapped I/O	**Isolated I/O**
Address Space	Shared with memory	Separate from memory
Instruction	No special I/O instructions. All memory access instructions can be used to access I/O registers.	Special I/O instructions
Kernel Mode	I/O operations only allowed in kernel mode	I/O operations only allowed in kernel mode
Distinction between memory access and I/O operations	Done using address decoder	Built-in distinction due to different opcodes

Appendix A: MIPS instruction set architecture

- **Instructions**
  - Arithmetic: add, addi, sub, mul
  - Load/Store: lw, lh, lhu, lb, lbu, sw, sh, sb, lui
  - Logic: sll, srl, and, andi, or, ori, nor
  - Control flow: beq, bne, j, jal, jr
  - Comparison: slt, slti, sltu, sltui
- **Pseudo-instructions**
  - move, blt
- **Registers** (32 in total) listed in numbering order
  - $zero, $at, $v0-$v1, $a0-$a3, $t0-t7, $s0-s7, $t8-$t9, $k0-$k1, $gp, $sp, $fp, and $ra
- **Instruction formats**
  - R-Format (add, sub, mul, sll, srl, and, or, nor, jr, slt, sltu), I-Format (addi, lw, lh, lhu, lb, lbu, sw, sh, sb, lui, andi, ori, beq, bne, slti, sltui), and J-Format (j, jal)

Appendix B: MIPS datapath (single cycle implementation)



Extra sheet