

University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2015

Instructors: Vladimir Stojanovic, John Wawrzynek

2015-11-10



CS61C MIDTERM 2



After the exam, indicate on the line above where you fall in the emotion spectrum between “sad” & “smiley”...

Last Name	Perfect
First Name	Peter
Student ID Number	
CS61C Login	cs61c-
The name of your SECTION TA (please circle)	Alex Austin Chris David Derek Eric Fred Jason Manu Rebecca Shreyas Stephan William Xinghua
Name of the person to your LEFT	
Name of the person to your RIGHT	
All the work is my own. I had no prior knowledge of the exam contents nor will I share the contents with others in CS61C who have not taken it yet. (please sign)	

Instructions (Read Me!)

This booklet contains 8 numbered pages including the cover page. It is followed by **an answer sheet**, which will be the only thing scanned into gradescope; write scratchwork there for credit. Finally, we have included a datapath diagram, a sheet of scratch paper, and the MIPS green sheet. **Please detach** the last four pages now and fill in your name, login, and SID on the answer sheet. After you finish the exam, turn in both the booklet and the answer sheet.

- Please turn off all cell phones, smartwatches, and other mobile devices. Remove all hats & headphones. Place your backpacks, laptops and jackets under your seat.
- You have 80 minutes to complete this exam. The exam is closed book; no computers, phones, or calculators are allowed. You may use two handwritten 8.5”x11” pages (front and back) of notes in addition to the provided green sheet.
- There may be partial credit for incomplete answers; write as much of the solution as you can. We will deduct points if your solution is far more complicated than necessary. Make sure your solution is on the answer sheet for credit.

	Q1	Q2	Q3	Q4	Q5	Total
Points Possible	15	20	20	15	10	80

Clarifications during the exam

Q2: Assume doBranch is 0 for bneqpc (doBranch only applies to standard branch instructions)

Q3.1: Find the # of stalls needed to resolve only those two instructions

Q4: Assume sizeof(int) returns 4, and that 'total' and 'i' are located in registers

Q4.4: Assume the cache is cold

Q5: Bias should be 31

Q5.3: Zero is not a positive number

Q1: Let's Adder All Up (15 points)

Consider the 4-bit adder shown to the right
It takes:

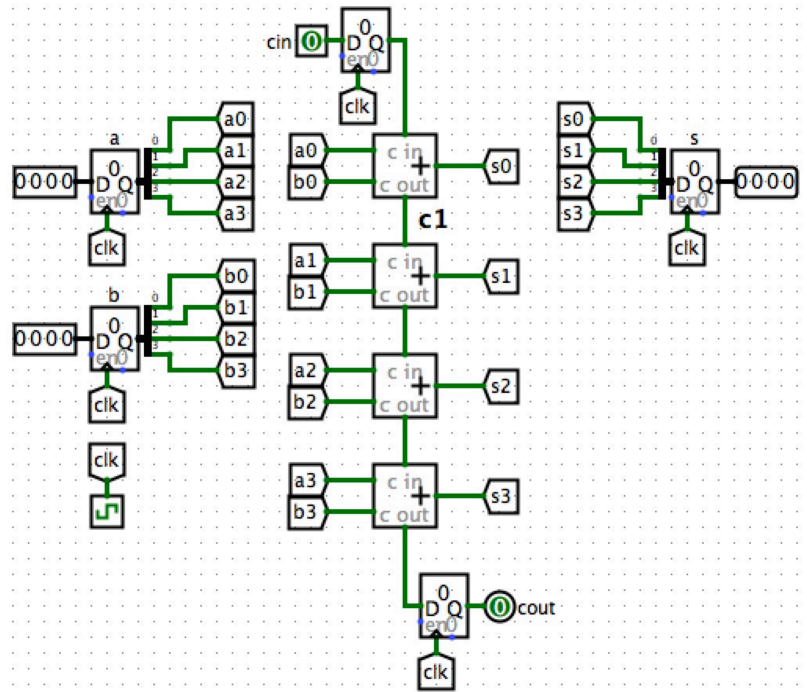
- a carry in (c_{in})
- two four-bit inputs:
 a with bits a_0, a_1, a_2, a_3
 b with bits b_0, b_1, b_2, b_3

Outputs:

- a carry out (c_{out})
- one four-bit output:
 s with bits s_0, s_1, s_2, s_3

Assume each adder has a delay of 10ns, and any registers have a clk-to-q, hold time, and setup time of 5ns. Assume the inputs are driven by registers, and outputs are registers as well.

Assume each adder has a delay of 10ns, and any registers have a clk-to-q, hold time, and setup time of 5ns. Assume the inputs are driven by registers, and outputs are registers as well.



1. Write Boolean formulas for s_0 and c_1 in terms of the inputs c_{in} , a_0 , and b_0 . You may use XOR as an operator in the Boolean formulas. Each formula should use as few operators as possible.

$$s_0 = a_0 \text{ XOR } b_0 \text{ XOR } c_{in}$$

$$c_1 = c_{in} * (a_0 \text{ XOR } b_0) + a_0 * b_0$$

2. What is the critical path delay of the circuit? Please include proper units in your answer.

$$50\text{ns} = \text{clk-to-q} + 4 \text{ adders} + \text{setup time}$$

3. What is the maximum clock frequency at which the circuit will function correctly? Please include proper units in your answer.

$$20 \text{ MHz} = 1/50 \text{ ns}$$

4. What is the maximum hold time the output registers could have at which the circuit would still function correctly?

$$15 \text{ ns}$$

The result for s_0 arrives in 15ns, so if it was greater, the hold time would be violated by the second set of inputs

Q2: Datapathology (20 points)

We want to implement a single-cycle MIPS CPU like the ones addressed in the course that can successfully execute the following instruction:

```
bneqpc $rt $rs IMM
```

RTL-esque description

```
if (R[$rs] != R[$rt]) {
    R[$rt] <- Mem[PC];
    PC <- R[$rs] + (Imm << 2); #HERE
} else {
    PC <- PC + 4
}
```

1. Give an example of an R[\$rs] and IMM that could potentially cause an error and explain why in two sentences or fewer.

R[\$rs] may not be a word-aligned address

Let's assume that the programmer/compiler always made sure that this was not an issue and in the instance where this occurred a trap was raised to debug the issue; do not worry about the potential issue.

For Q2.2, you will be modifying the CPU in order to implement **bneqpc**. You must also finish filling out the control signal table in Q2.3.

Important observations:

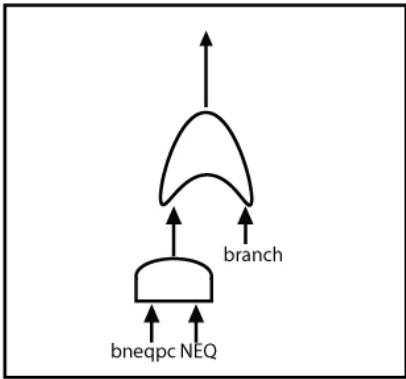
- Assume that the current state of the CPU does **not** account for an implementation of **bneqpc** at all.
- You are given that the ALU generates a signal "NEQ" that is high when two inputted arguments are not equal and low otherwise.
- Use the signal "bneqpc" as a control signal that is high when the passed-in instruction is **bneqpc**.

Make sure that the abilities of the original MIPS CPU are still preserved. Hint: Look through all answers before attempting to solve the question. **Also, take a look at the given control signals in Q2.3 when working through Q2.2.**

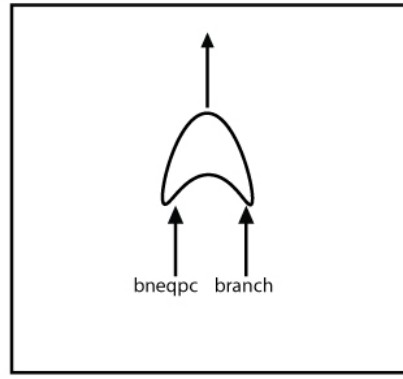
After the answer sheet is an incomplete datapath diagram that you can detach. Your job is to find the correct circuit fragments from the choices below to complete the datapath.

2. Your choices for the datapath boxes I-V are shown below. Select the best option that will allow you to implement **bneqpc**. If you find multiple valid solution combinations, choose the one that uses the **fewest number of extra hardware units overall**.

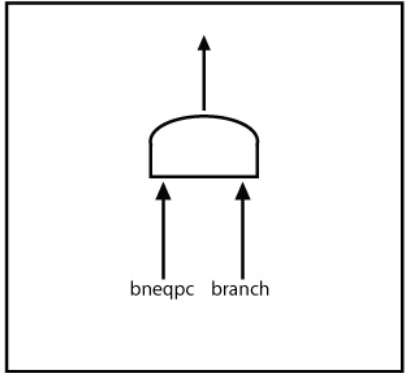
I. If none of the answers below are sufficient, feel free to draw your answer in D.



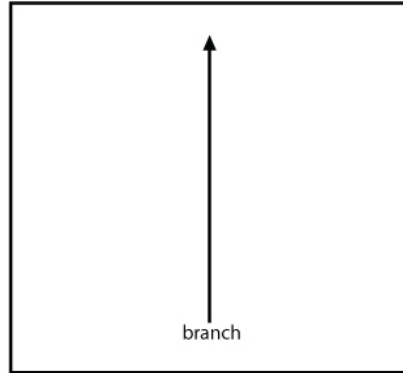
A



B



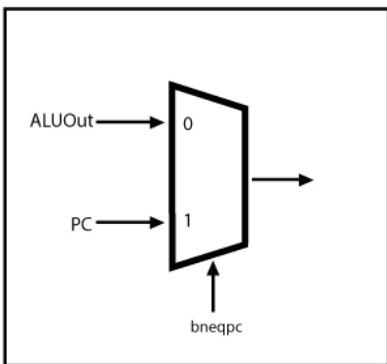
C



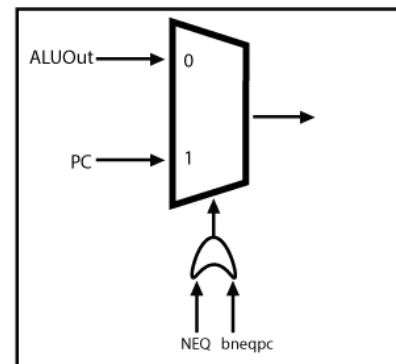
D

The branch signal does not account for bneqpc and should not be changed

II. If none of the answers below are sufficient, feel free to draw your answer in D.

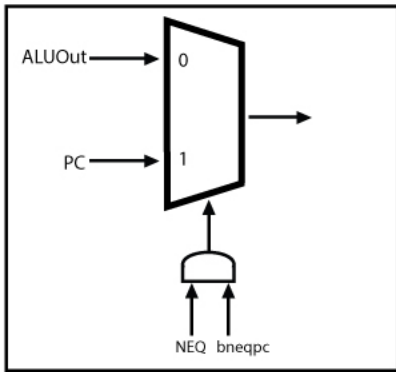


A

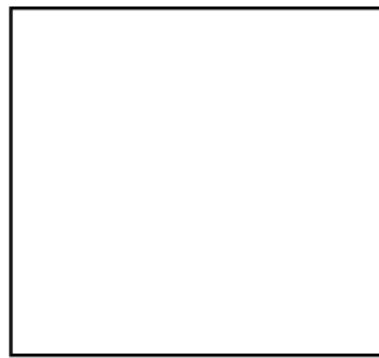


B

all instructions take **ALUOUT**, **EXCEPT** for **bneqpc** which takes **PC**. We do not need to worry about **NEQ** because that can be accounted for solely in question 5

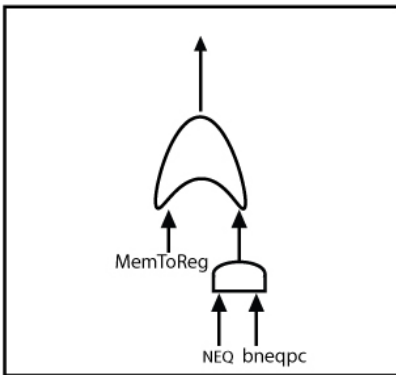


C

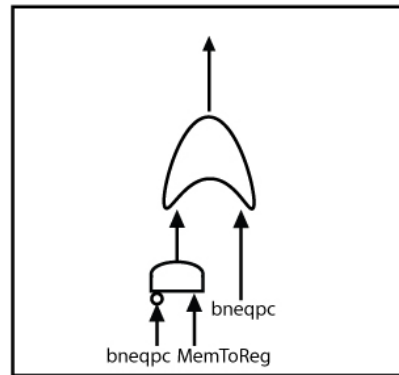


D

III. One of the answers is below.

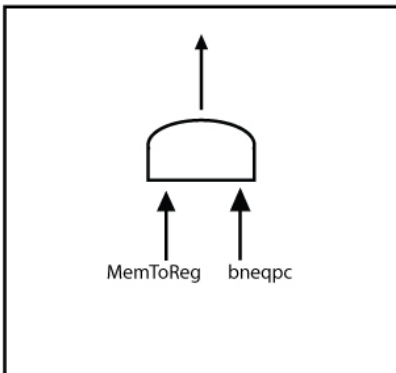


A

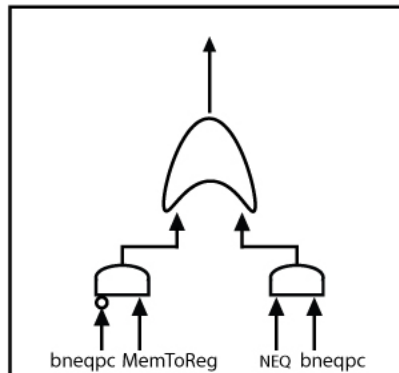


B

The main changes we need to make is how we choose upon writing to a register (RegWr) and modifying the PC (RelAddr, JumpAddr, or bneqpcAddr). With those implemented, accounting for the NEQ condition at this stage is superfluous. Furthermore, MemToReg is not guaranteed to be high for bneqpc. We also don't want to interfere with the normal influence of MemToReg for other instructions.

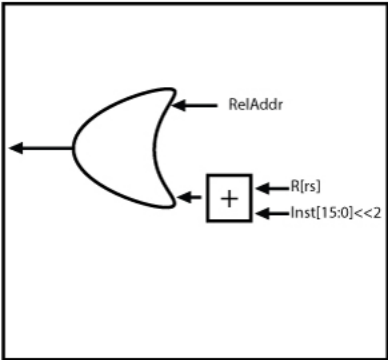
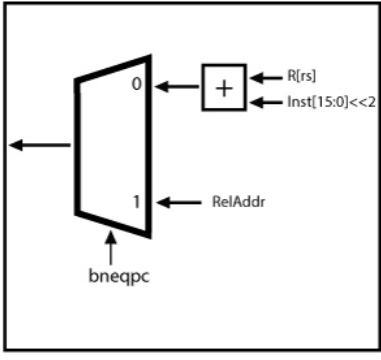
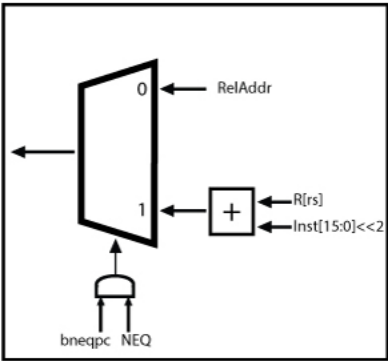
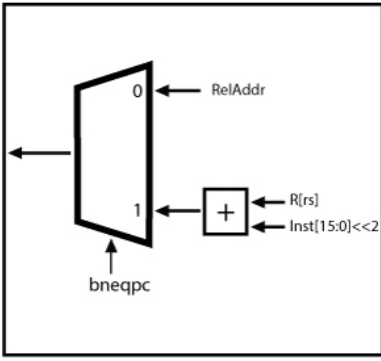


C

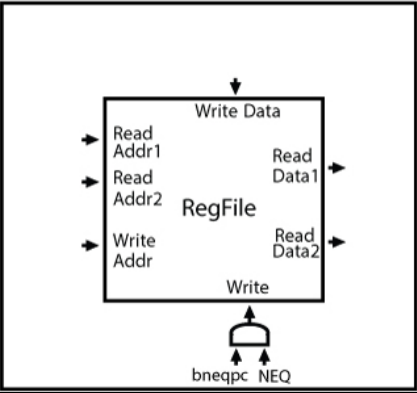
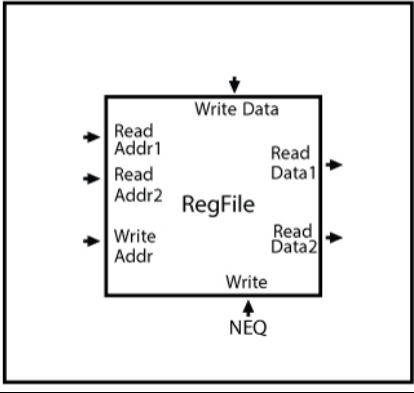


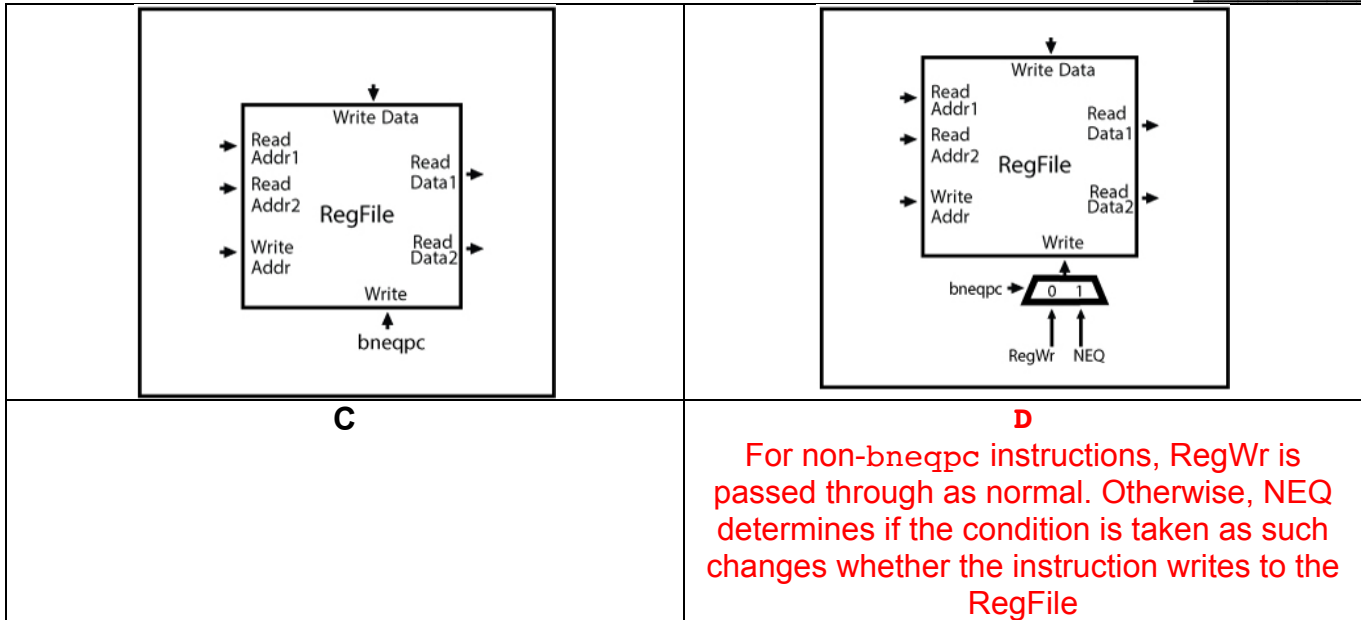
D

IV. One of the answers is below.

 <p>A</p>	 <p>B</p>
 <p>C</p>	 <p>D</p>
<p style="color: red;">We need a way to choose between the RelAddr (PC+4 or the branch address) as well as the bneqpc address. We only want to take the bneqpc address if the condition is filled. Also, we don't have to worry about after this step because the jump signal is low.</p>	

V. If none of the answers below are sufficient, feel free to draw your answer in D.

 <p>A</p>	 <p>B</p>
-----------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------



3. Fill in the control signals in the answer document, some are provided for you. 0 = low, 1 = high, XXX = doesn't matter.

Jump	Branch	RegDst	ExtOp	ALUSrc	ALUCtr	MemWr	MemToReg	RegWr
0	xxx	0	SIGN	0	Equals	0	XXX	XXX

Q3: Hazardous Conditions (20 points)

Assume that we have a standard 5-stage pipelined CPU with **no forwarding**. Register file writes can happen before reads, in the **same clock cycle**. We also have comparator logic that begins at the beginning of the decode stage and **calculates the next PC by the end of the decode stage**. For now, assume there is **no branch delay slot**. The remainder of the questions pertains to the following piece of MIPS code:

	Instructions	Cycle									
		1	2	3	4	5	6	7	8	9	10
0	start: addu \$t0 \$t1 \$t4	IF	D	EX	MEM	WB					
1	addiu \$t2 \$t0 0		IF	D	EX	MEM	WB				
2	ori \$t3 \$t2 0xDEAD			IF	D	EX	MEM	WB			
3	beq \$t2 \$t3 label				IF	D	EX	MEM	WB		
4	addiu \$t2 \$t3 6					IF	D	EX	MEM	WB	
5	label: addiu \$v0 \$0 10						IF	D	EX	MEM	WB
6	syscall										

1. For each instruction dependency below (the line numbers are given), list the type of hazard and the length of the stall needed to resolve the hazard. If there is no hazard, circle "no hazard".

- 0 → 1: data hazard, 2 cycles
- 0 → 3: no hazard
- 1 → 3: data hazard, 1 cycle
- 2 → 3: data hazard, 2 cycles
- 3 → 4: control hazard, 1 cycle

For the following questions, assume that our CPU now has forwarding implemented as presented in class and in the book.

2. Which of these instruction dependencies would cause a pipelining hazard?

- A. 2 → 3: **ori \$t3 \$t2 0xDEAD → beq \$t2 \$t3 label**
- B. 2 → 4: ori \$t3 \$t2 0xDEAD → addiu \$t2 \$t3 6
- C. 2 → 5: ori \$t3 \$t2 0xDEAD → addiu \$v0 \$0 10
- D. 3 → 4: **beq \$t2 \$t3 label → addiu \$t2 \$t3 6**
- E. None of the above

3. If we were given a **branch delay slot**, which instruction would reduce the most amount of pipelining hazards if moved into the branch delay slot? If all instructions are equally beneficial, or no instruction removes any hazards, write “nop” as your answer.

addiu \$v0 \$0 10

Q4: Cache Rules Everything Around Me (15 points)

You are given a MIPS machine with a single level of **2KiB direct-mapped** cache with **512B cache blocks**. It has **1MiB of physical address space**.

The function foo is ran on the system with a **cold** cache and as the only process:

```
#define ARRAY_LEN 4096
#define STEP_SIZE 64

// A starts at 0x10000
// B starts at 0x20000

foo( int* A, int* B ) {
    int total = 0;
    for ( int i = 0; i < ARRAY_LEN; i += STEP_SIZE ) {
        total += A[ i ];
        total -= B[ i ];
    }
}
```

1. Calculate the number of Tag, Index, and Offset bits for this cache.

Tag: 9

Index: 2

Offset: 9

2. Calculate the hit percentage for this cache after running foo.

0%

3. The cache is now cleared and the code is run again. This time, **A** and **B** are pointing to the same array, which starts at **0x10000**. Calculate the new hit percentage.

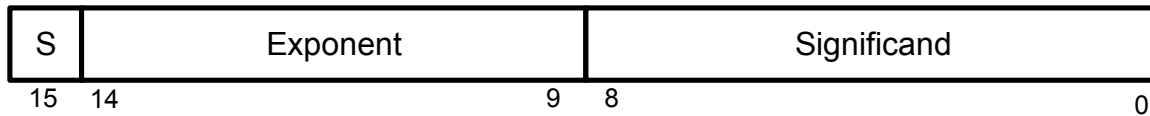
75%

4. Assume **A** and **B** starts once again at **0x10000** and **0x20000**. What is the new hit percentage if we ran foo on a fully associative cache, with all other parameters staying the same?

50%

Q5: Don't Let Your Mind Float Away Now (10 points)

Consider the following 16-bit representation for floating point numbers:



Bits per field:

- Sign: 1
- Exponent: 6
- Significand: 9
- Everything else follows the IEEE standard 754 for floating point, except in 16 bits

Bias: -31

1. Convert -15.125 into floating point. Write your answer in hexadecimal.

0xB5B8

2. What is the value of the largest odd number that can be represented by the above floating point representation?

$2^{10} - 1$

3. How many positive, real numbers can be represented?

$2^{15} - 2^9 - 1$

