COMP 122/L Final Practice Exam (Answers)

Final Exam Topics:

- The use of memory on MIPS (with the `lw` and `sw` instructions) and branches (conditionals and loops)
- Truth tables
- Circuits
- Boolean formulas
- De Morgan's laws
- Karnaugh maps (K-maps), including those with *don't cares*

In addition to this review, you should also review:

- Related Labs (Labs 4 - 6)
- All the handouts

This review itself is not intended to be comprehensive

1.) Consider the following `.data` section of a MIPS program:

```
.data
array:
  .word 7, 2, 4
```

Finish this MIPS program so that it prints out every element of this array, WITHOUT using a loop. As a hint, you'll need to use different offsets in your `lw` instructions. Don't forget to terminate the program.

```
.text
main:
  la $t0, array
  li $v0, 1

  lw $a0, 0($t0)
  syscall
  lw $a0, 4($t0)
  syscall
  lw $a0, 8($t0)
  syscall


  li $v0, 10
  syscall
```

2.) Consider the following `.data` section of a MIPS program:

```
.data
array:
  .word 7, 2, 4, 8
copy:
  .word 0, 0, 0, 0
```

Finish this MIPS program so that `copy` will contain a copy of the contents of `array`. You MUST use a loop. Don't forget to terminate the program.

```
.text
main:
  la $t0, array # $t0: pointer to current source element
  la $t1, copy  # $t1: pointer to current destination element
  li $t2, 4     # $t2: number of elements remaining

loop:
  lw $t3, 0($t0) # $t3: holds current element
  sw $t3, 0($t1)
  addiu $t0, $t0, 4
  addiu $t1, $t1, 4
  addiu $t2, $t2, -1
  bne $t2, $zero, loop

  li $v0, 10
  syscall
```

3.) Consider the following C-like code:

```
int array[] = {4, 8, 9, 1, 0, 5};
for (int index = 0; index < 6; index += 2) {
  int temp = array[index];
  array[index] = array[index + 1];
  array[index + 1] = temp;
}
```

This code started to be translated to MIPS assembly as follows:

```
.data
array:
  .word 4, 8, 9, 1, 0, 5
```

Complete the translation of this code. Don't forget to terminate the program.

```
# Equivalent code with while loop:
# int array[] = {4, 8, 9, 1, 0, 5};
# int index = 0;
# while (index < 6) {
#   int temp = array[index];
#   array[index] = array[index + 1];
#   array[index + 1] = temp;
#   index += 2;
# }

.text
main:
  la $t0, array   # pointer to current element
  li $t1, 0       # current count (index)

loop:
  sltiu $t2, $t1, 6
  beq $t2, $zero, after_loop
  lw $t3, 0($t0)    # $t3: temp; temp = array[index]
  lw $t4, 4($t0)    # $t4 = array[index + 1]
  sw $t4, 0($t0)    # array[index] = $t4
  sw $t3, 4($t0)    # array[index + 1] = temp;
  addiu $t0, $t0, 8  # go two elements past
  addiu $t1, $t1, 2
  j loop

after_loop:
  li $v0, 10
  syscall
```

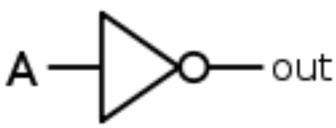
4.) What component is shown below?

An OR gate.

5.) What component is shown below?

An AND gate.

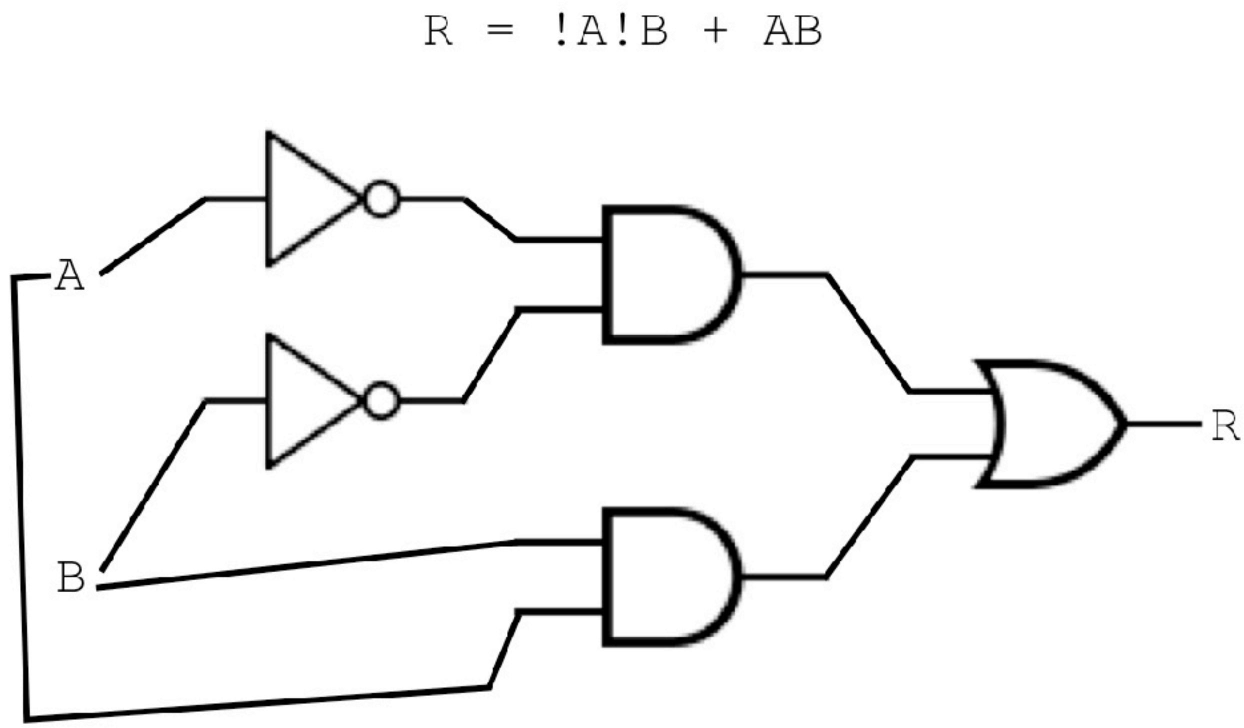
6.) What component is shown below?



A NOT gate.

7.) Draw the circuit corresponding to the following sum-of-products equation:

```
R = !A!B + AB
```



8.) Consider the following sum-of-products equation:

```
R = !ABC + ABC + A!B!C
```

8.a.) Write the equation as a truth table.

A	B	C	R
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

8.b.) Simplify it using a Karnaugh map.

A \ BC	00	01	11	10
0	0	0	1	0
1	1	0	1	0

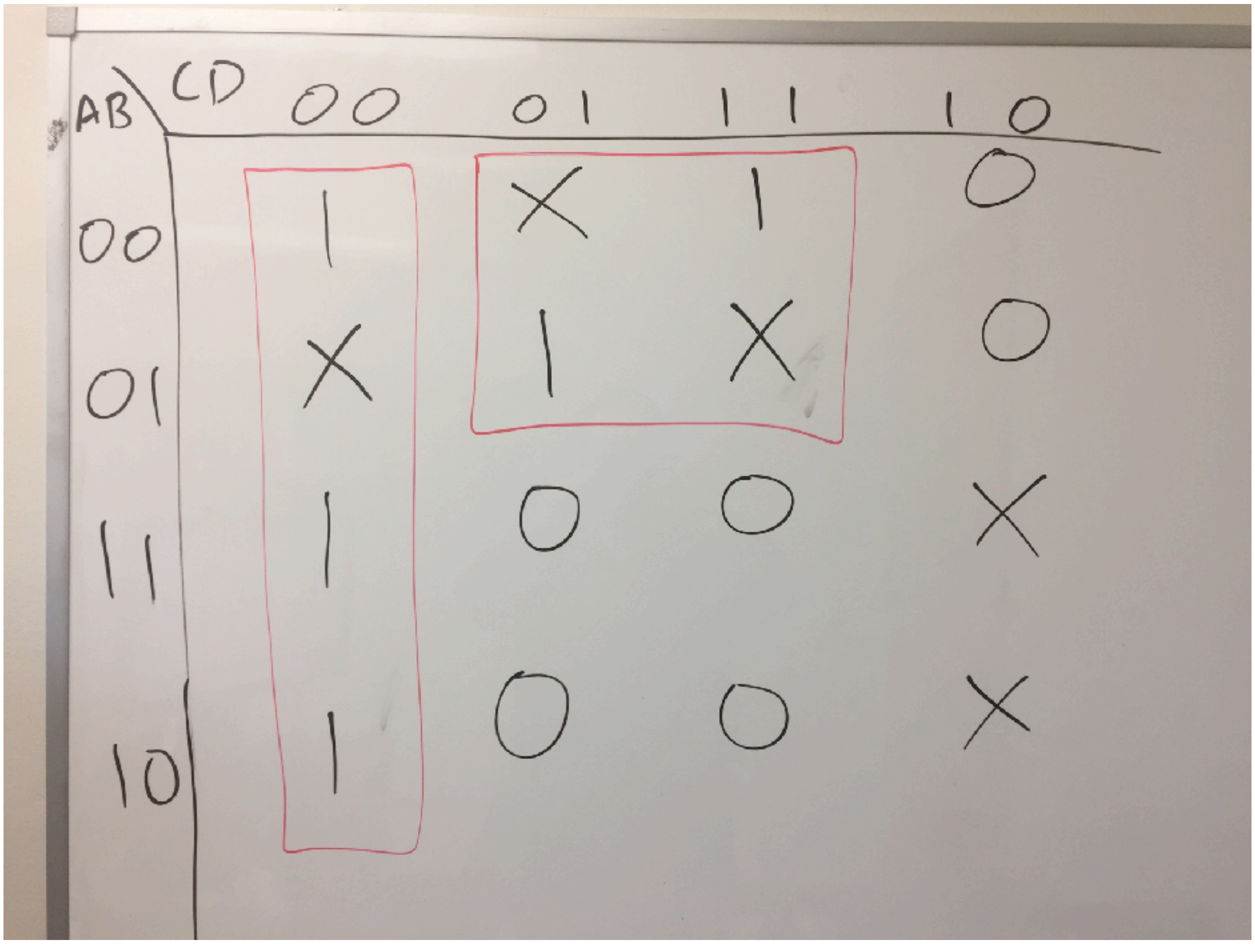
R = A!B!C + BC

A	B	C	D	U
0	0	0	0	1
0	0	0	1	X
0	0	1	0	0
0	0	1	1	1
0	1	0	0	X
0	1	0	1	1
0	1	1	0	0
0	1	1	1	X
1	0	0	0	1
1	0	0	1	0
1	0	1	0	X
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	X
1	1	1	1	0

9.) Consider the truth table below, which includes *don't cares*:
9.a.) Write out the unoptimized sum-of-products equation corresponding to this truth table. As a hint, *don't cares* can be skipped over.

U = !A!B!C!D + !A!BCD + !AB!CD + A!B!C!D + AB!C!D

9.b.) Using a K-map, derive an optimized equivalent sum-of-products equation that exploits *don't cares* where appropriate.

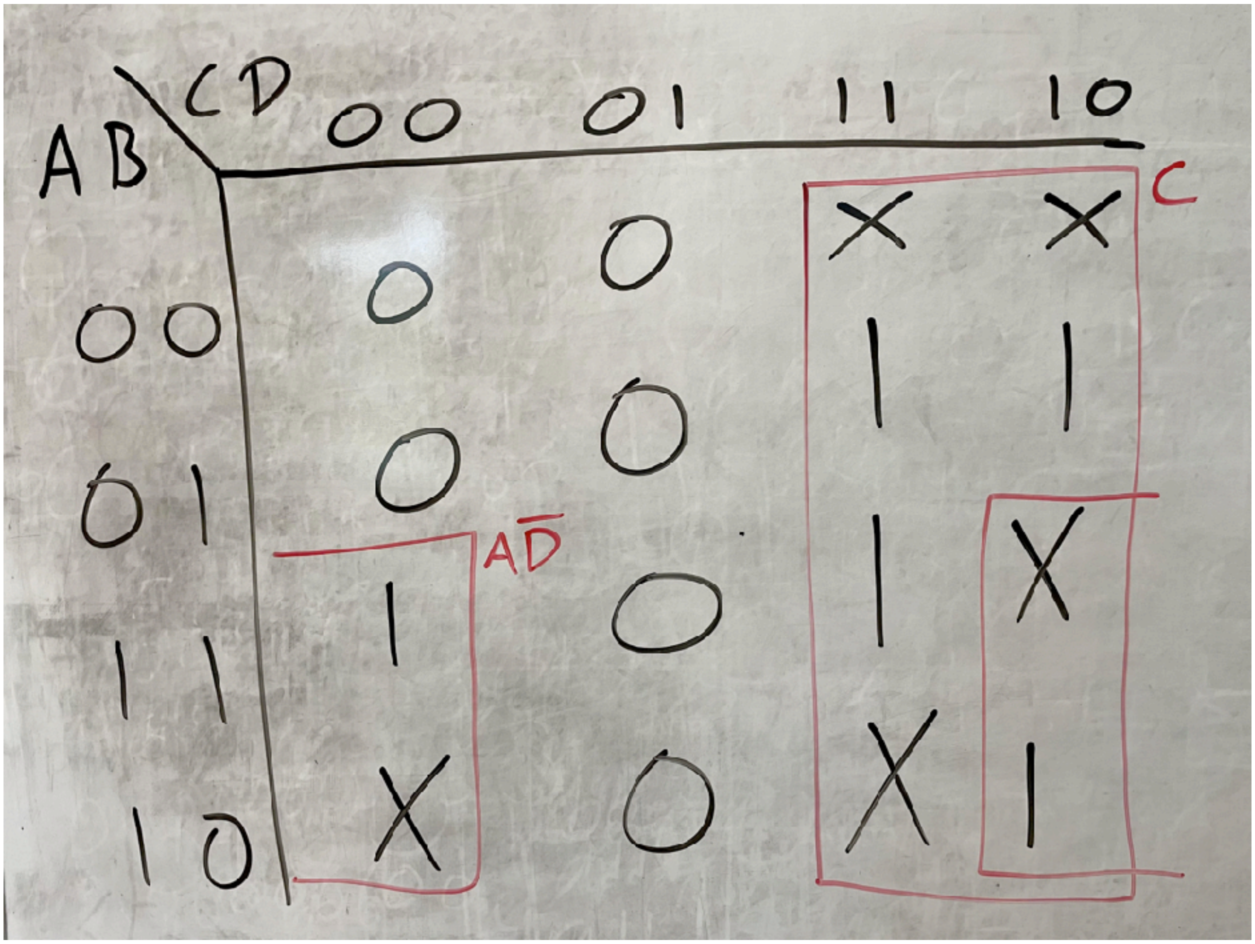


U = !C!D + !AD

10.) Consider the truth table below, which includes *don't cares*:

A	B	C	D	Output
0	0	0	0	0
0	0	0	1	0
0	0	1	0	X
0	0	1	1	X
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	X
1	0	0	1	0
1	0	1	0	1
1	0	1	1	X
1	1	0	0	1
1	1	0	1	0
1	1	1	0	X
1	1	1	1	1

Using a K-map, derive an optimized equivalent sum-of-products equation that exploits *don't cares* where appropriate.



Output = A!D + C