EEL 4712
Midterm 3 – Spring 2017
VERSION 1

Name: ______________________

UFID: ______________________

Sign here to give permission for your test to be returned in class, where others might see your score:

____

IMPORTANT:
• Please be neat and write (or draw) carefully. If we cannot read it with a reasonable effort, it is assumed wrong.
• **As always, the best answer gets the most points.**

COVER SHEET:

Problem#:	Points
1 (4 points)	
2 (4 points)	
3 (4 points)	
4 (8 points)	
5 (4 points)	
6 (4 points)	
7 (4 points)	
8 (4 points)	
9 (4 points)	
10 (4 points)	
11 (4 points)	
12 (8 points)	
13 (10 points)	
14 (4 points)	
15 (4 points)	
16 (20 points)	
17 (6 points)	6

Total:

Regrade Info:

1) (4 points) FPGA synthesis replaces tristates with what type of component?

Mux

2) (4 points) What is the name of the technology that uses virtual reconfigurable architectures implemented atop an FPGA to improve productivity?

Overlays (intermediate fabrics, supernets also ok)

3) (4 points) When the behavior of a circuit demonstrates some amount of randomness, it is likely that the output of a FF is:

`metastable`

4) a. (2 points) When synchronizing a single bit across clock domains, what type of synchronizer should you use?

Dual flop

b. (2 points) When synchronizing multiple bits across clock domains and don't care about throughput, what type of synchronizer should you use?

Handshake or mux recirculation

c. (2 points) When synchronizing multiple bits across clock domains and throughput is important, what type of synchronizer should you use?

FIFO

d. (2 points) In what situation can dual-flop synchronizers be used to synchronize multiple bits?

When only one bit changes

5) (4 points) How many bits are in each MIPS instruction?

32

6) (4 points) How many registers are in the MIPS register file?

32

7) (4 points) What is the purpose of the data stored into register 31 during a jump and link instruction?

Function return address

8) (4 points) Briefly explain the difference between the behavior of an r-type and i-type instruction.

R-type instructions use 2 registers as input.
I-type instructions use 1 register and a 16-bit immediate value for input

9) (4 points) Briefly explain the difference between a jump and branch instruction.

A jump is unconditional and uses an absolute address instead of an offest

10) (4 points) What is the purpose of the HI and LO registers in the MIPS datapath?

Multiplication produces a 64-bit product, which requires two 32-bit registers: HI and LO

11) (4 points) Describe the functionality that occurs during the instruction-fetch stage. Keep your description high level. E.g., you do not need to mention control signals.

IR = RAM[PC]
PC = PC + 4

12) (8 points) Write MIPS assembly code that does the following behavior. Assume the a[] starts at address 0x1000 and the b[] array starts at 0x2000. Use $r0-$r31 for registers. Make two columns if necessary.

```
for (int i =0; i < 100; i++) {
      a[i] = b[i] + 4;
}
```

```
        addiu $r1, $r0, $r0      // i =0
COND:
        subiu $r2, $r1, 100      // compare i and 100
        bgez $r2, DONE           // if i >= 100, don't loop
        lw $r3, 0x2000($r1)      // $r3 = b[i]
        addiu $r3, $r3, 4        // b[i] + 4
        sw $r3, 0x1000($r1)      // a[i] = b[i] + 4
        addiu $r1, $r1, 1        // i++
        j COND
DONE:
```

13) (10 points) Create a memory initialization file for the following assembly code. Add comments as necessary. Put a small space between different instruction fields to make it easier to read.

```
LOOP:
       beq $r1, $r3, DONE
       addiu $r5, $r4, 0x1000
       lw $r1, 0($r5)
       j LOOP
DONE:
       j DONE
```

```
Depth = 256;
Width = 32;
Address_radix = hex;
Data_radix = bin;
% Program RAM Data %
Content
Begin

-- LOOP
00: 000100 00001 00011 0000000000000011        -- beq
01: 001001 00100 00101 0001000000000000        -- addiu
02: 100011 00101 00001 0000000000000000        -- lw
03: 000010 00000000000000000000000000          -- j LOOP
-- DONE
04: 000010 00000000000000000000000100          -- j DONE
```

// NOTE: IF YOU USED MULTIPLES OF 4 FOR YOUR INSTRUCTION ADDRESSES, LIKE THE PROVIDED MIF FILES, THEN THE ADDRESS OF THE FINAL JUMP ADDRESS WILL NEED TO CHANGE. THE BRANCH OFFSET DOES NOT NEED TO CHANGE. <u>THIS DOES NOT APPLY TO CLASSES AFTER SPRING 2017.</u> SEE BELOW:

```
-- LOOP
00: 000100 00001 00011 0000000000000011        -- beq
04: 001001 00101 00100 0001000000000000        -- addiu
08: 100011 00101 00001 0000000000000000        -- lw
0C: 000010 00000000000000000000000000          -- j LOOP
-- DONE
10: 000010 00000000000000000000010000          -- j DONE
```

```
End;
```

14) (4 points) Given a solution space with the following implementations, which of the solutions are ***not*** Pareto optimal? If they are all Pareto optimal, state that.

   a. Area: 1000 LUTs, Time: 18s
   b. Area: 2000 LUTs, Time: 19s
   c. Area: 3000 LUTs, Time: 8.5s
   d. Area: 4000 LUTs, Time: 16s
   e. Area: 5000 LUTs, Time: 20s

15) (4 points) During design-space exploration, you are considering two implementations:

   1. Area: 5000 LUTs, Time: 3s
   2. Area: 4000 LUTs, Time: 3s

Is the existence of implementation 2 sufficient to prove that implementation 1 is not Pareto optimal? Explain your answer.

Yes, implementation one is bigger with the same performance, so it is never a better options.

No is an acceptable answer under the assumption that there are other metrics not being considered (power, energy, cost, time-to-market, etc.)

16) a. (5 points) For the following code, create a schedule for the provided datapath. Ignore muxes, registers, and other glue logic. Like the examples in class, assume that address calculations are done *without* using the specified resources (i.e., address calculations cost nothing). Do not change the code. List any assumptions.

```
for (int i=0; i < 10000; i++) {
   a[i] = b[i]*10 + b[i+1]*20 + b[i+2]*30 + b[i+3]*40;
}
```

Datapath
1 multipliers
1 adders
1 comparator
1 memory for b[] (can read 1 elements/cycle)
1 memory for a[] (can write 1 element/cycle)

c = b[i]*10
d = b[i+1]*20
e = b[i+2]*30
f = b[i+3]*40

0) i=0
1) i < 10000, load b[i]
2) load b[i+1], c
3) load b[i+2], d
4) load b[i+3], e, c+d
5) f, (c+d)+e
6) (c+d+e)+f
7) store a[i], i++

b. (2 points) What is the execution time in total cycles based on your schedule from part a? Show your work.

7 cycles/iteration * 10000 iterations + 1 cycle = 70001 cycles

c. (5 points) Create a new schedule for a datapath with 4 multipliers, 2 adders, 1 comparator, and a memory for b[] that can read 4 inputs/cycle.

0) i=0
1) i < 10000, load b[i….i+3]
2) c,d,e,f
3) c+d, e+f,
4) (c+d)+(e+f)
5) store a[i], i++

d. (2 points) What is the execution time of the schedule from part c?

5*10000+1 = 50001 cycles

c. (4 points) For a pipelined implementation of the datapath in part c, what is the approximate execution time in total cycles?

5 + 9999 = 10004 cycles

d. (2 points) Give two reasons why pipelining should be considered before applying loop unrolling.

1) **At the same level of performance, pipelining uses less area**
2) **Pipelining requires lower peak memory bandwidth than loop unrolling**

17) (6 points) Free points for being the first class to do the MIPS lab.