Name: ______________________

First Name: ______________________

Student ID: ______________________

2nd session examination

Design of Digital Circuits WS2016

(252-0014-00L)

Srdjan Capkun, Frank K. Gürkaynak

Examination Rules:

1. Written exam, 90 minutes in total.
2. No books, no calculators, no computers or communication devices. Six pages of handwritten notes are allowed.
3. Write all your answers on this document, space is reserved for your answers after each question. Blank pages are available at the end of the exam.
4. Put your Student ID card visible on the desk during the exam.
5. If you feel disturbed, immediately call an assistant.
6. Answers will only be evaluated if they are readable.
7. Write with a black or blue pen (no pencil, no green or red color).
8. Show all your work. For some questions, you may get partial credit even if the end result is wrong due to a calculation mistake.
9. Points for every part are indicated in the exam. They should correspond to the expected difficulty of the questions. Consider this when allocating your time.

Question:	1	2	3	4	5	6	Total
Points:	5	16	24	10	8	12	75
Score:							

This page intentionally left blank

1. (a) (2 points) For the following four numbers given in decimal or hexadecimal notation, write the corresponding binary number using the indicated format.

$(-48)_{10}$ using 8-bit sign magnitude:	$(1011\,0000)_2$
$(155)_{10}$ using 8-bit unsigned:	$(1001\,1011)_2$
$(-28)_{10}$ using 8-bit two's complement:	$(1110\,0100)_2$
$(BA)_{16}$ using 8-bit unsigned:	$(1011\,1010)_2$

(b) (3 points) State whether the following statements about the binary representation of numbers are *true* or *false*. Give **brief** explanations for the statements that are *false*.

- Both two's complement and sign/magnitude representation can be used to represent negative numbers in binary.

  Solution: True, however it is more difficult to design arithmetic circuits that work with sign/magnitude format. Still they are used.

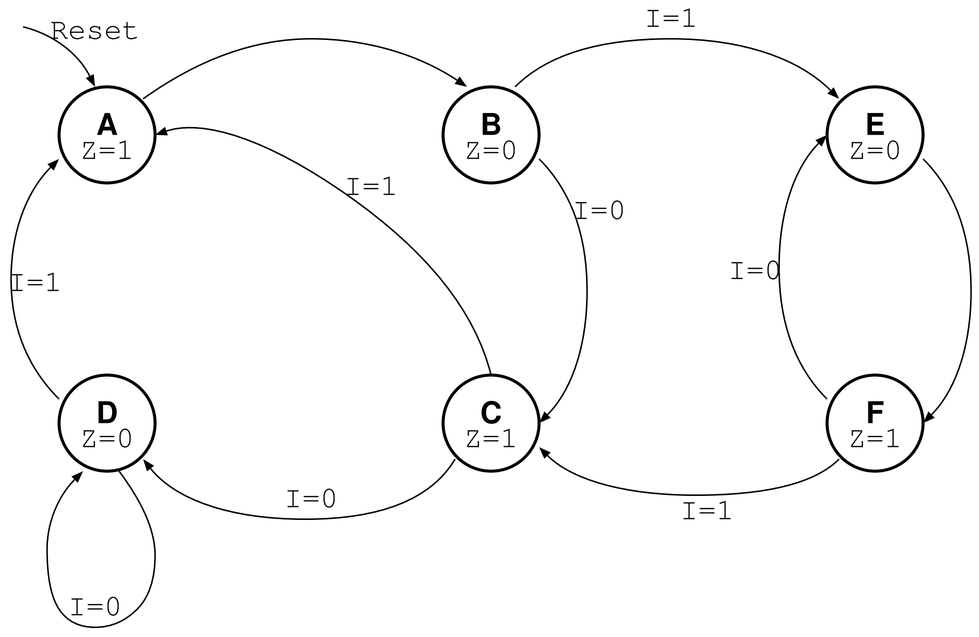
- Using N bits it is possible to represent 2^N different numbers when a sign/magnitude number system is used.

  Solution: False, 0 is represented twice.

- While there are methods to represent both positive and negative integers, it is not possible to represent fractions or real numbers using binary numbers.

  Solution: False, fixed and floating point number systems can be used to represent such numbers.

2. Consider the following state diagram of an FSM with 1-bit input (I) and 1-bit output (Z).



The state has been coded using a 3-bit vector $S = S_2\, S_1\, S_0$ according to the following table:

State			
name	S_2	S_1	S_0
A	0	0	0
B	0	0	1
C	0	1	0
D	0	1	1
E	1	0	0
F	1	0	1

(a) (1 point) Is this a Moore or Mealy type FSM? Briefly explain.

Solution: Moore, outputs depend only on the present state and nothing else.

(b) (6 points) Fill in the following state transition table that determines the next state vector $N = N_2\,N_1\,N_0$ based on the current state S and the input I.

State				**Input**	**Next State**			
name	S_2	S_1	S_0	I	name	N_2	N_1	N_0
A	0	0	0	X	B	0	0	1
B	0	0	1	0	C	0	1	0
B	0	0	1	1	E	1	0	0
C	0	1	0	0	D	0	1	1
C	0	1	0	1	A	0	0	0
D	0	1	1	0	D	0	1	1
D	0	1	1	1	A	0	0	0
E	1	0	0	X	F	1	0	1
F	1	0	1	0	E	1	0	0
F	1	0	1	1	C	0	1	0

Note that there are different ways of writing this table to represent the same result.

(c) (3 points) Write the Next State Equations from the table you have filled above using either *Sum of Products (SOP)* or *Product of Sums (POS)* form. **Do not spend time minimizing the equations, this will be next question**.

Solution:

$N_0 = \overline{S_2}\,\overline{S_1}\,\overline{S_0} + \overline{S_2}\,S_1\,\overline{S_0}\,\overline{I} + \overline{S_2}\,S_1\,S_0\,\overline{I} + S_2\,\overline{S_1}\,\overline{S_0}$

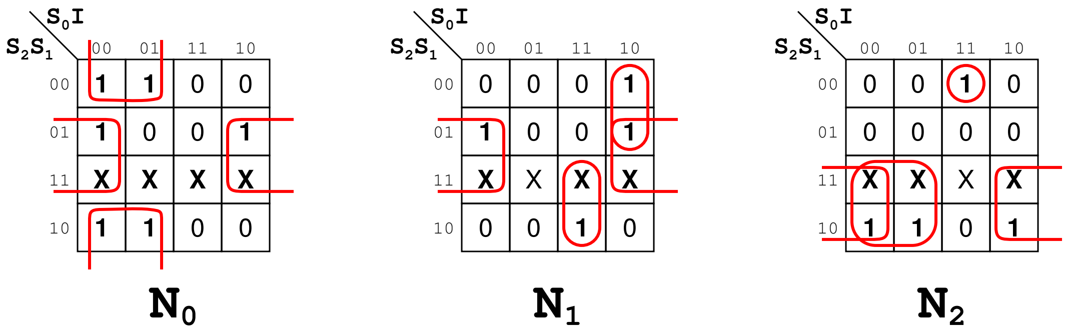
$N_1 = \overline{S_2}\,\overline{S_1}\,S_0\,\overline{I} + \overline{S_2}\,S_1\,\overline{S_0}\,\overline{I} + \overline{S_2}\,S_1\,S_0\,\overline{I} + S_2\,\overline{S_1}\,S_0\,I$

$N_2 = \overline{S_2}\,\overline{S_1}\,S_0\,I + S_2\,\overline{S_1}\,\overline{S_0} + S_2\,\overline{S_1}\,S_0\,\overline{I}$

(d) (6 points) Minimize the Next State Equations from the previous part. Note that the FSM requires only six states. This means that there are several State (S) / Input(I) combinations for which the outputs can be treated as *Don't Care*, which should help minimizing the boolean equations.
Hint: Consider using Karnaugh diagrams to solve this problem.

Solution:

It is important to note that for $S_2\,S_1 = 11$ the next state N can be taken as $N_2\,N_1\,N_0 = XXX$. This can simplify the Boolean equations significantly. It is best to use a Karnaugh map to find the simplifications.



$N_0 = \overline{S_1}\,\overline{S_0} + S_1\,\overline{I}$

$N_1 = S_1\,\overline{I} + S_2\,S_0\,I + \overline{S_2}\,S_0\,\overline{I}$

$N_2 = S_2\,\overline{S_0} + S_2\,\overline{I} + \overline{S_2}\,\overline{S_1}\,S_0\,I$

3. This question is about SRAMs. Subquestions can be answered independently. If you have skipped the previous part, and think you need to make some assumptions, please clearly state your assumptions.

   (a) (2 points) For the design of a microprocessor you are given a pre-designed SRAM with 16-bit wide data inputs (`din`) and outputs (`dout`), fourteen address bits(`addr`), as well as the write enable (`we`) signal that is set to 1 when data is written to memory. The Verilog secription of this module is given below:

```
module sram (input [15:0] din,
             input [13:0] addr,
             input we,
             input clk,
             output [15:0] dout);

   // description of the module
endmodule
```

   How many bytes can be stored in this memory (in Kilobytes)?

   Solution: Each address stores $16/8 = 2$ bytes. There are 14 address bits, that allow 2^{14} addresses, which equals to $2^4 \times 2^{10} = 16 \times 1024 = 16k$ addresses. In total this makes $16k \times 2bytes = 32kBytes$.

   (b) (2 points) Using the memory `sram` described in 3.a we want to build a larger memory called `bigram`, that will have the following interface.

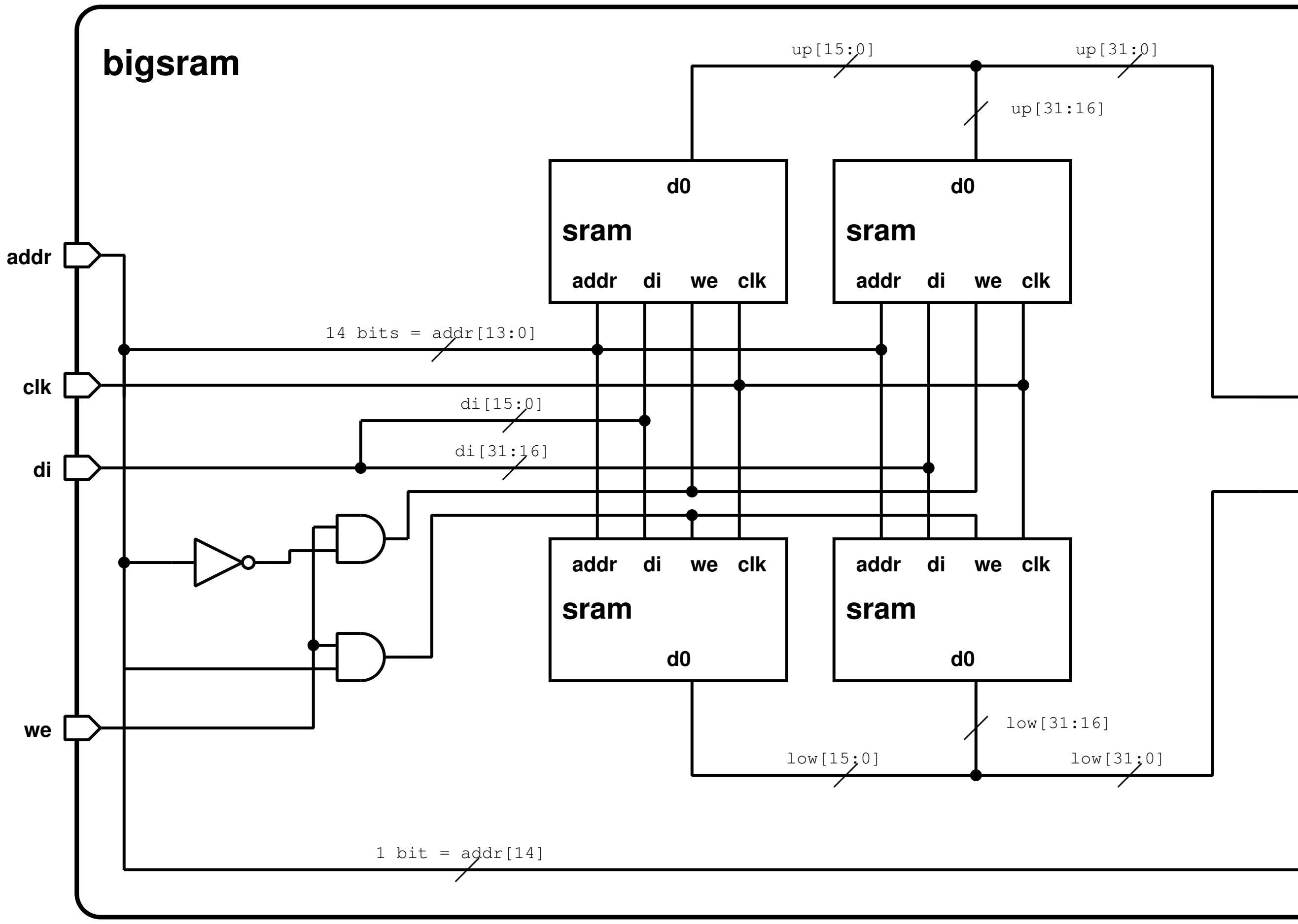
```
module bigram (input [31:0] din,
               input [14:0] addr,
               input we,
               input clk,
               output [31:0] dout);

   // description of the module
endmodule
```

   How many instances of the `sram` module do you need to create to build `bigram`? Why?

   Solution: The `bigram` stores twice the number of bits, 32 instead of 16. So we will need to use two `sram` instances. Then the `bigram` also uses one more bit for the address, doubling the number of locations. So we will need to double the number of `sram` instances once again, for a total of 4.

(c) (10 points) Draw a block diagram of the `bigram` and show in detail how it can be obtained by combining sufficient number of `sram` instances. To make your life easier, consider what needs to be done for the data input, data output and write enable separately.



(d) (10 points) Write the Verilog code that assembles `bigram` from sufficient number of instances of `sram`

```
module bigram (input [31:0] din,
               input [14:0] addr,
               input we,
               input clk,
               output [31:0] dout);

// declare signals if necessary
  wire [31:0]  up, low;
  wire         we_up, we_low;

// instantiate sram instances
  sram up0  (.din(din[15:0]),  .dout( up[15:0]),
             .addr(addr[13:0]), .we(we_up),  .clk(clk) );
  sram up1  (.din(din[31:16]), .dout( up[31:16]),
             .addr(addr[13:0]), .we(we_up),  .clk(clk) );
  sram low0 (.din(din[15:0]),  .dout(low[15:0]),
             .addr(addr[13:0]), .we(we_low), .clk(clk) );
  sram low1 (.din(din[31:16]), .dout(low[31:16]),
             .addr(addr[13:0]), .we(we_low), .clk(clk) );

// write enable
  assign we_up  = addr[14] ? we : 0;
  assign we_low = addr[14] ? 0 : we;

// select output
  assign dout = addr[14] ? up : low;

endmodule
```

4. In this section, you will be given a task and two code snippets in MIPS assembly language. You will have to decide which of the code snippets can be used for the task. **For all the questions assume the following initial values**:

Registers:

Register	Value
`$s0`	`0x0000 00FF`
`$s1`	`0x0000 0004`
`$s2`	`0x0000 0008`
`$s3`	`0x0000 000C`

Memory:

Address	Value
`0x0000 00000`	`0x0000 FF00`
`0x0000 00004`	`0x0000 00FF`
`0x0000 00008`	`0xFFFF FFF7`
`0x0000 0000C`	`0x1234 5678`

(a) (2 points) Set the content of the register `$t1` to `0x0000 1234`:

(A)

```
    lw  $t1, 0xC($0)
    srl $t1, $t1, 4
```

(B)

```
    xor $t1, $t1, $t1
    addi $t1, $0, 0x1234
```

☐ none ☐ A [✓ B] ☐ Both A and B

A is wrong since it is shifting the value by 4 bits. It should shift the value by 16 bits instead.

(b) (2 points) Starting from the address `0x0000 4000` write all zeroes to 1024 consecutive memory locations (until `0x0000 5000`):

(A)

```
      addi $s0, $s0, 0x1000
LOOP: sw   $0,  0x4000($s0)
      addi $s0, $s0, -1
      bne  $s0, $0,  LOOP
```

(B)

```
      addi $s0, $s0, 0x4000
      addi $s1, $s0, 0x1000
      addi $s2, $0,  1
LOOP: sw   $0,  $s1
      sub  $s1, $s1, $s2
      bne  $s0, $s1, LOOP
```

☐ none ☐ A ☐ B [✓ **Both A and B**]

(c) (2 points) Add all the numbers from 0 to 255:

(A)

```
      lw    $s1, $s0
      xor   $s0, $s0, $s0
LOOP: add   $s0, $s0, $s1
      addi  $s1, $s1, -1
      bne   $s1, $0, LOOP
```

(B)

```
      addi $s1, $0, 255
      lw   $s0, $0
LOOP: addi $s0, $0, $s1
      addi $s1, $s1, -1
      beq  $s1, $0, DONE
      j    LOOP
DONE:
```

■ **none** ☐ A ☐ B ☐ Both A and B

A is wrong since the lw command loads the memory pointed by the register s0, not the value stored in s0. B is wrong since addi takes an immediate number as the third argument, not a register.

(d) (2 points) Jump to subroutine STOP if only the 4th bit from the right (representing 2^3) of the byte written at address `0x0000 0020` is 1 while other bits are all 0. Otherwise continue with the program at CONT:

(A)

```
      lw   $s0, 0x20($0)
      srl  $s0, $s0, 3
      addi $s1, $0, 1
      beq  $s0, $s1, CONT
      j    STOP
CONT: ...
STOP: ...
```

(B)

```
      addi $s0, $0, 0x20
      lw   $s1, $s0
      lw   $s2, 0x8($0)
      and  $s3, $s1, $s2
      beq  $s3, $0, CONT
      jal  STOP
CONT: ...
STOP: ...
```

■ **none** ☐ A ☐ B ☐ Both A and B

A looks ok, but it would also work if other bits (higher than 4) are one as well, the jump is not to a subroutine, and the condition is inverse. B would be okay if it were bne instead of beq.

(e) (2 points) Save the two registers `$s0` and `$s1` to the stack:

(A)

```
      sw    $sp, $s0
      addi  $sp, $sp, -4
      sw    $sp, $s1
```

(B)

```
   addi $sp, $sp, -8
   sw   $s0, 8($sp)
   sw   $s1, 4($sp)
```

☐ none ☐ A ■ **B** ☐ Both A and B

A is incorrect since the stack pointer itself is overwritten instead of the actual memory pointed by the stack pointer.

---

5. (a) (3 points) Briefly explain **two** advantages of a multi-cycle architecture when compared to a single-cycle architecture.

Solution:

- In a single-cycle architecture, all instructions are given 1-cycle to execute, therefore the slowest instruction determines the speed of the processor.
- In a multi-cycle processor, instructions are broken down into smaller pieces, decreasing the cycle time. Simpler instructions can be executed faster, reducing the average cycle time.
- A single cycle processor, needs multiple instances of memories, and adders which may be quite large. A multi-cycle processor can share these resources, using only a single memory and ALU. This reduces the area

(b) (5 points) For each of the following statements about microarchitectures, write if their TRUE or FALSE. If they are FALSE, explain why.

- In a pipelined architecture, a given instruction is executed faster than in a single-cycle architecture.

  Solution: FALSE, a given instruction runs even slightly slower, due to the overhead, but the throughput increases

- Control and Data Hazards can not occur in single or multi-cycle architectures.

  Solution: TRUE, they only occur when instructions are executed at the same time in parallel, like in pipelined architectures.

- The higher the Cycles per Instruction (CPI) of a micro-architecture, the faster it will finish its operation.

  Solution: FALSE, all other things being equal, a high CPI will lower the execution speed.

- A single-cycle architecture has less control overhead than a multi-cycle architecture

  Solution: TRUE, a multi-cycle architecture has more resources to be shared, and there is overhead for the sequential processing.

6. (a) (5 points) Consider a 2-way set associative cache (N) with a memory word size of 4 bytes, and a block size of 64 words (b). What is the capacity (C) of this cache in bytes, if you want to have 128 sets (S) in the cache?

Solution: A cache of Capacity C contains B = C/b blocks where b is the block size. In our example b is 64 x 4bytes = 256 Bytes. An N-way set associative cache has S= B/N sets. In our case for S=128, and N=2, we need 256 blocks. So the capacity is 256 x 256 Bytes = 64 kBytes.

(b) (7 points) Explain what the function of the *tag* is in a cache. How many bits would a cache for a 32-bit word addressed MIPS architecture need to store the tags for the cache configuration described in part a)

Solution: The cache only stores a small part of the memory. Only one part of the address is used to address the data in the cache. Since many addresses can map to the same location, the rest of the address has to be stored with the data everytime a cache block is updated. This part of the address is called *tag* and is stored alongside the data in the cache.

A 32-bit MIPS processor uses 32 bits for addressing, and due to the word addressing the last of these address bits are always zero, leaving 30 effective address bits. There are 64 words in a block of our cache, so 6 bits ($2^6 = 64$) will be needed to select the words in a block. There are 128 sets in our cache, so 7 bits ($2^7 = 128$) of the address will be used to select the set. This leaves 17 bits (32 -2 -6 -7) for the tag in each way. There are two ways and 128 sets so 256 x 17 == 4352 bits will be needed to store the tags for this cache.