University of California, Berkeley – College of Engineering

Department of Electrical Engineering and Computer Sciences

Fall 2016 Instructors: Randy Katz, Bernhard Boser 2016-11-1

☹CS61C MIDTERM 2☺

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

Last Name	
First Name	
Student ID Number	
CS61C Login	`cs61c-`
*The name of your **SECTION** TA and time*	
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)***	

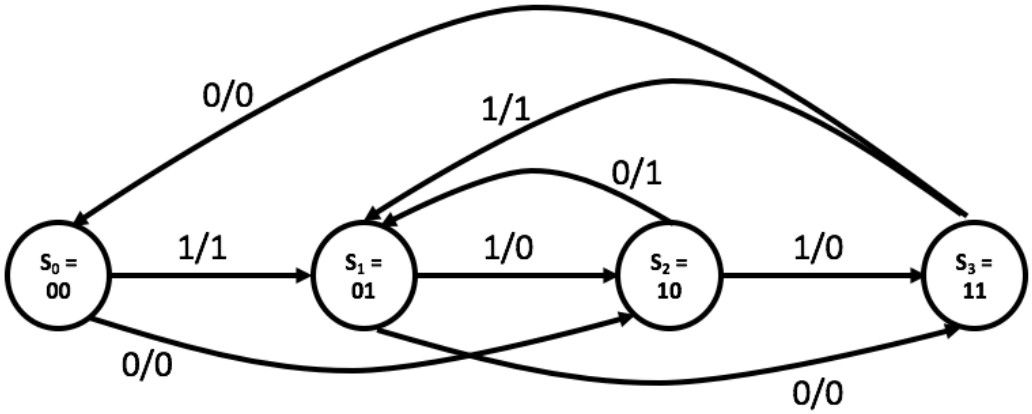
Instructions (Read Me!)

- This booklet contains 9 numbered pages of text including the cover page.
- 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 **one** handwritten 8.5"x11" page (front and back) crib sheet in addition to the MIPS Green Card, which we will provide.
- 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. When we provide a blank, please fit your answer within the space provided.
- Points are assigned by the approximate time to answer the question, 1 point = 1 minute. Pace yourself, and at least attempt every question for partial credit.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Total
Points Possible	8	8	14	2.5	3.5	12	12	60

Q1: Finite State Machine (8 points)

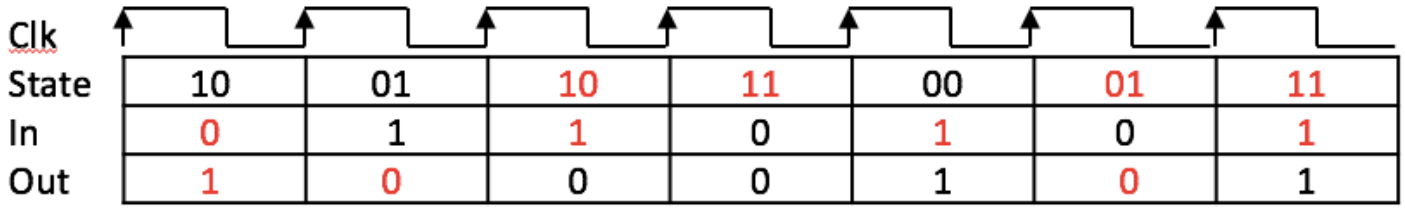
Answer the questions below for the finite state machine in this diagram:



1. Complete the truth table shown below. (2 points)

Input		Output	
State	**In**	**State**	**Out**
S_0 = 00	0	S_2 = 10	0
00	1	01	1
01	0	11	0
01	1	10	0
10	0	01	1
10	1	11	0
11	0	00	0
11	1	01	1

2. Fill in the blanks in the diagram below. (3 points.)



3. The finite state machine is required to operate at a frequency f_{clk} = 5GHz. The finite state machine is realized with combinatorial logic with delay t_c = 120ps and flip-flops with hold and clock-to-Q times of t_{hold} = 50ps and t_{clk2Q} = 70ps, respectively.

   What is the maximum value of the flip-flop setup time t_{setup} that allows the finite state machine to operate at a clock frequency of up to f_{clk} = 5GHz? Suggestion: draw a complete timing diagram. (3 points)

   $t_{setup} \leq$ _____200-120-70=10ps_____ (state the unit of your result).

Q2: Pipelining (8 points)

Compare two pipeline implementation options A and B with 4 and 7 stages, respectively.

1. The logic delays of the pipeline stages are as follows:

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
Option A	250 ps	180 ps	400 ps	200 ps			
Option B	200 ps	150 ps	250 ps	250 ps	200 ps	150 ps	180 ps

What are the maximum clock rates for the two implementations? (2 points)

Option A f_{s_max} = _____1/0.4ns = 2.5GHz_____ (include the unit with your result)
Option B f_{s_max} = _____1/0.25ns = 4GHz_____ (include the unit with your result)

2. The table below states the operation of each pipeline stage:

	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
Option A	IF/ID	EXE	MEM	WB			
Option B	IF	ID	EXE-1	EXE-2	EXE-3	MEM	WB

Compared to the MIPS CPU, option A merges IF and ID in a single stage, while option B splits EXE over three pipeline stages. Registers and memory are written to in the first half of the cycle and read during the second half of the cycle (same as MIPS) but there are *no forwarding paths*. How many instructions are executed after an **`add`** and a **`lw`** instruction, respectively, before the new register values are available? (4 points)

	# nops after `add`	# nops after `lw`
Option A	4	4
Option B	2	2

3. Calculate the number of instructions executed per second for each implementation for the specified f_s (these are *not necessarily the correct results for part 1*) and CPI. (2 points)

1/CPI * 1/f = 1/CPI * Cycles / sec

	f_s	CPI	Instructions per second
Option A	3 GHz	1.5	$(1/1.5)(1/((1/3)*10^{-9}))= 2*10^9$
Option B	5 GHz	2.0	$(1/2.0)(1/((1/5)*10^{-9})) = 2.5*10^9$

Q3: Pipeline Hazards (14 points)

The goal of the problem is to increase the execution speed of the code below by eliminating as many stalls and useless operations (**nop**'s in the branch delay slots) as possible. The code runs on a 5-stage pipelined MIPS CPU with forwarding with the characteristics discussed in lecture.

Note: the branch delay slot is not hidden, i.e. **nop**'s in the code below are always executed regardless of the branch decision.

a) Indicate stalls in the code below with arrows right after the instruction where the stall occurs. The following is not necessarily correct and only used as an example. (2 points)

stall →	addiu $s0, $s0, 1
	addiu $s1, $s1, 4
	addiu $s2, $s2, 4

b) How many cycles does it take to execute the entire code sequence below, including stalls and **nop**s? (2 points)

N1 = 19 or 20 (depends on if you observed the stall between sltiu and beq) cycles

c) Assuming all stalls and nops can be eliminated, by how many cycles does the execution time decrease? (2 points)

N2 = 4 or 5 (depends on if you observed the stall between sltiu and beq) cycles

d) Rewrite the code below, eliminating as many stalls and **nop**s as possible. The improved code must store the same results to memory, but register values may differ between the two versions when **exit** is reached. (8 points)

There were multiple accepted solutions for this part. Extra credit was given to students who observed the stall between sltiu and beq (not necessary for full credit).Students were provided credit if their solution was optimal and the results stored to memory after executing the code remained the same.

Original Code Sequence

loop:	sltiu $t0, $s0, 100
	beq $t0, $0, exit
	nop
	lw $t0, 0($s1)
➔	addiu $t0, $t0, 1
	lw $t1, 0($s2)
➔	addu $t0, $t0, $t1

Improved Code Sequence

loop:	sltiu $t0, $s0, 100
	beq $t0, $0, exit
	addiu $s0, $s0, 1
	lw $t0, 0($s1)
	lw $t1, 0($s2)
	addiu $t0, $t0, 1
	addu $t0, $t0, $t1

	sw $t0, 0($s2)
	addiu $s0, $s0, 1
	addiu $s1, $s1, 4
	addiu $s2, $s2, 4
	beq $0, $0, loop
	nop
exit:	

	sw $t0, 0($s2)
	addiu $s1, $s1, 4
	beq $0, $0, loop
	addiu $s2, $s2, 4
exit:	

Q4: Locality (2.5 points)

Choose the **single best answer** that describes the locality characterized by the indicated C programming pattern. Assume "well-written" code. (0.5 points each)

i. Sequencing of Instructions in a loop

○ Temporal ○ Spatial ○ Both ○ Neither

ii. Subroutine prologue and epilogue

○ Temporal ○ Spatial ○ Both ○ Neither

iii. String copy

○ Temporal ○ Spatial ○ Both ○ Neither

iv. Nested If-Then-Else Processing

○ Temporal ○ Spatial ○ Both ○ Neither

v. Two Dimensional Matrix Multiply

○ Temporal ○ Spatial ○ Both ○ Neither

Q5: AMAT (3.5 points)

Fill in the following parts. Recall that AMAT = Hit Time + Miss Rate * Miss Penalty.

a.) Assume Hit Time is 1 cycle and the miss penalty is 100 cycles. What must the miss rate be to achieve an AMAT of 2 cycles? (1 point)

```
2 = 1 + x * 100
x = 0.01
```

_____

b.) As in (a.) but now assume the miss penalty is 800 cycles. What is the miss rate needed to achieve an AMAT of 2 cycles? (1 point)

```
2 = 1 + x * 800
x = 0.00125
```

_____

c.) Assume a two level cache where the L1 hit time is as in (a.) and the hit time in the L2 cache is 2 cycles and the miss penalty is 800 cycles. Further assume an L1 miss rate of 0.1. Show how you calculate the L2 miss rate to achieve an AMAT of 2 cycles. What is the L2 miss rate you calculated? (1.5 points)

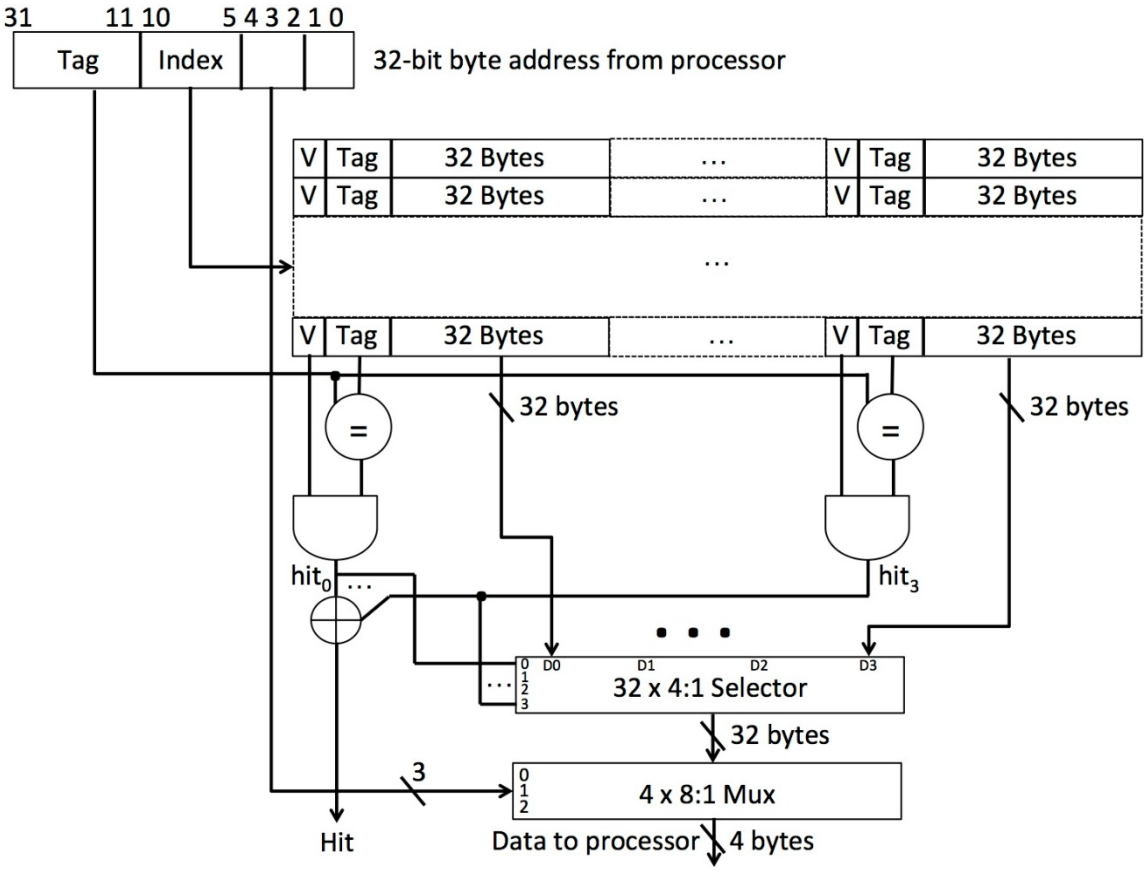
```
AMAT = L1 hit + L1 miss rate *
          (L2 hit time + L2 miss rate * L2 miss penalty)

2 = 1 + .1 (2 + x * 800)
2 = 1 + .2 + x * 80
.8 = x * 80
x = 0.01
```

_____

Q6: Reverse Engineering a Cache (12 points)

You are given the sketch of a cache design below:



Note: a 4:1 Selector has four control inputs labeled 0, 1, 2, 3 and four data inputs D0, D1, D2, D3. It connects D_i to the output when the ith control signal is true. Assume that **at most one** control input is true at any time. The selector function is undefined when none of the control inputs are true.

Answer the following questions about the cache above.

a.) What is the block size of the cache in bytes? (1 point) 32 Bytes

b.) What is the number of blocks in this cache? (1 point) 256

c .) What is the total data capacity of the cache in bytes? (2 points) 8192 Bytes

d .) What is the associativity of the cache? (2 points) 4-way

e .) Is this a write-through or write-back cache? (2 points) Write-through

f .) What is the total number of valid bits in the cache? (2 points) 256

g .) What is the total number of tag bits in the cache? (2 points) 256 * 21 = 5376

Q7: Program Performance in Caches (12 points)

You are given a snippet of MIPS assembly code that copies a null terminated character string from one location to another.

```
# $t0 has the address of the string source
# $t1 has the address of the string destination

StrCpy:
      lb    $t2, 0($t0)         # $t2 gets src char
      sb    $t2, 0($t1)         # store into dest char
      beq   $t2, $zero, Exit    # done when last char is null
      addiu $t0, $t0, 1         # point to next char of src
      addiu $t1, $t1, 1         # point to next char of dest
      j   StrCpy

Exit:
```

Assume the processor is **unpipelined**, and has a unified instruction and data cache that is direct mapped with 128 bytes and word-sized blocks. Assume random replacement, write through, and no write allocate. The `StrCpy` code is located at (byte) memory address 0 and the source and destination strings at (byte) memory addresses 128_{10} and 256_{10} respectively. Show your work to receive full credit.

a.) For just the sequence of instructions shown above, how many memory references are there for the loop iteration that copies a non-null character? (1 point)

Six instruction refs and two data refs = **8 memory refs**

b.) For just the sequence of instructions shown above, how many memory references (i.e. access to memory) are there for the loop iteration that copies the null character and exits the loop? Do not count the reference to the instruction at label `Exit`. (1 point)

Three instruction refs and two data refs = **5 memory refs**

For the following questions, assume the source string is four bytes long (3 bytes plus null terminator). You can write the miss rate as the fraction # of misses/# of processor memory references.

c.) What is the total number of memory references to execute this snippet of code? (1 point)

Three iterations x 8 memory references + one iteration of 5
= **29 total**

(Questions continued on the other side)

d.) What is the cache miss rate? (3 points)

First iteration: 6 instruction misses, 1 data miss, since no write allocate, store is thru to memory, and not stored in cache. Load knocks out first instruction. The store should count as a miss too. 8 misses total

Second iteration: miss on first instruction, hit on five instructions, miss on load data (instruction fetch kicked it out) and miss on store: 3 misses total.

Third and fourth iterations are the same.

Total misses: 8 + 3 + 3 + 3 = **17 misses/29 memory references = 0.59**

Common mistakes were 13/29 which assumed no write allocate never tried accessing cache.

e.) Now consider a two-way set associative cache of the same capacity and block size. What is the cache miss rate now? (3 points)

No conflict between first instruction and source string address. Stores still count as misses though.

Total misses: 8 + 1 + 1 + 1 = **11 misses/29 memory references = 0.38**

Common mistakes were 7/29 which assumed no write allocated never tried accessing cache.

f.) Now consider a two-way-set-associative cache, but organized as a write-back cache with write-allocate. The replacement strategy is random with a preference for clean data blocks. What is the miss rate now? (3 points)

First iteration: 6 instruction misses, one load data miss, store data written to cache but not memory. 8 misses total (need to load the store data block).

Second iteration can be 1 miss or 2 misses based on replacement policy.

Third and fourth iterations will be two misses per iteration (instruction ref knocks out load but store stays in the cache as it is dirty; there should just be conflict between the instruction and the load data).

8 + 2 + 2 + 2 = **14 misses/29 references = 0.48**
OR
8 + 1 + 2 + 2 = **13 misses/29 references = 0.45**