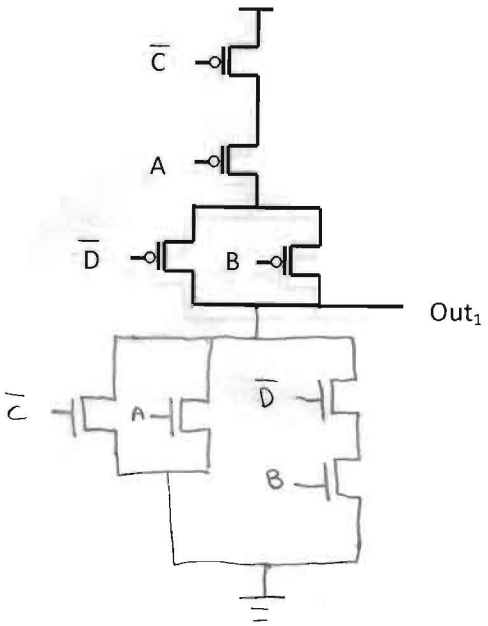
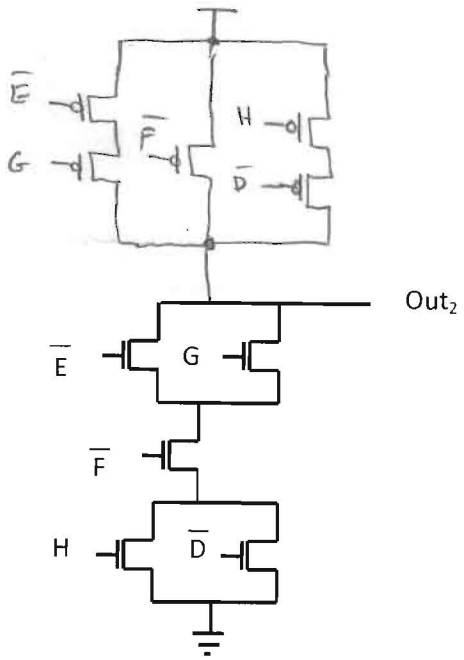
ECE2020 Final Exam Summer 2013 GTL
July 31, 2013 Name: KEY

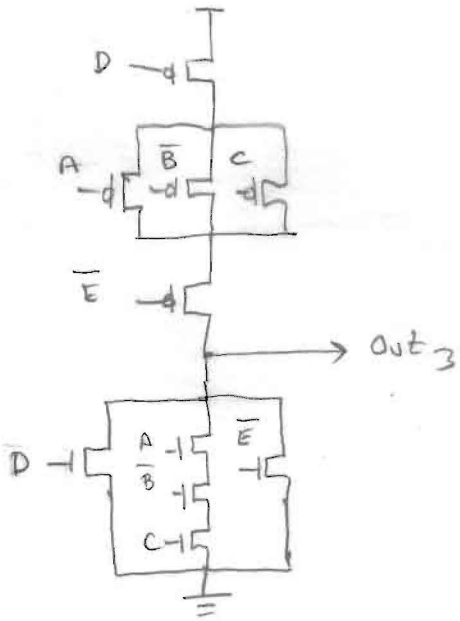
Show your work for any possible partial credit. 100 possible points, 12 exam pages plus a figure.

High Level Language		
Assembly Language		
Instruction Set		
Memory	Data Path	Controller
Storage	Functional Units	State Machines
Building Blocks		
Gate		
Switches and Wires		

1) (6 Points) Three incomplete circuits are shown below. Complete each circuit by adding the needed switching network so the output is pulled high or low for all combinations of inputs (i.e., no floats or shorts). Complete each circuit (pull down, pull up, or both) and write the expression if one is not given. Assume both inputs and complements are available.





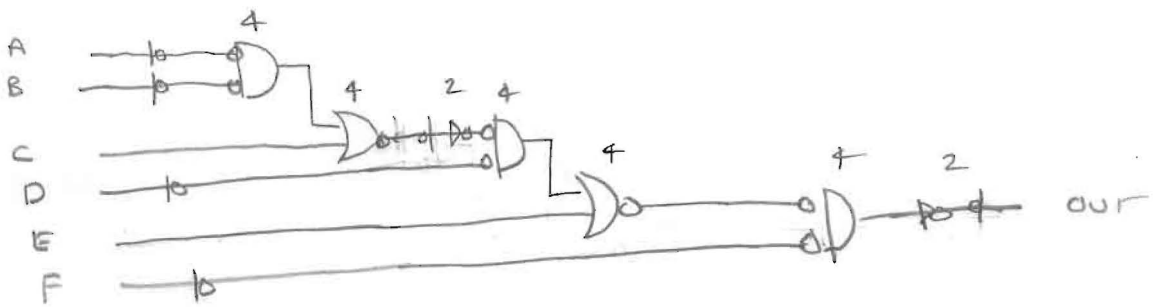


Out$_1$ = $C\overline{A}(D+\overline{B})$ Out$_2$ = $E\overline{G} + F + \overline{H}D$ Out$_3$ = $\overline{D}\ (\overline{A} + B + \overline{C})\ E$

High Level Language		
Assembly Language		
Instruction Set		
Memory	Data Path	Controller
Storage	Functional Units	State Machines
Building Blocks		
Gate		
Switches and Wires		

2) (5 points) Implement the following expression using only NOR gates and inverters. Then determine the number of switches required. Use proper mixed logic notation. Do not modify the expression. Do not assume compliments of inputs are available.

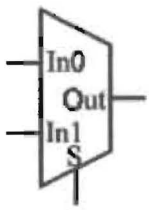
Out = $(\overline{\overline{A} \cdot \overline{B} + C})\ \overline{D} + E)\ \overline{F}$



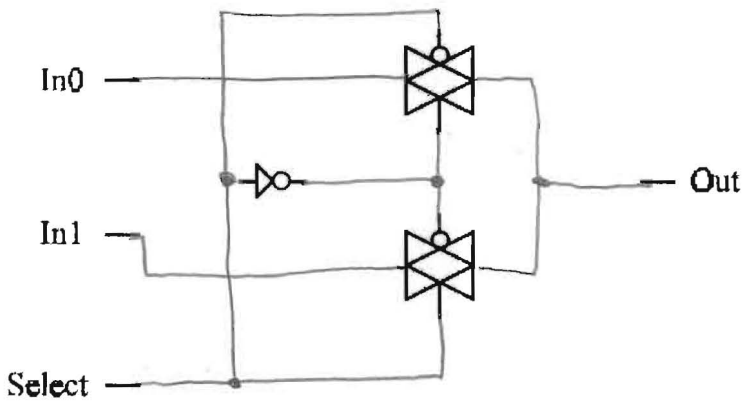
2

Number of switches 5x4 + 2x2 = 24

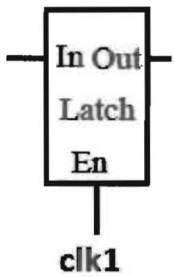
High Level Language		
Assembly Language		
Instruction Set		
Memory	Data Path	Controller
Storage	Functional Units	State Machines
Building Blocks		
Gate		
Switches and Wires		



3) (5 points) Using only the devices below, implement a two-to-one mux on the diagram below. (The block diagram of a 2 to 1 mux is show above just as an FYI only).

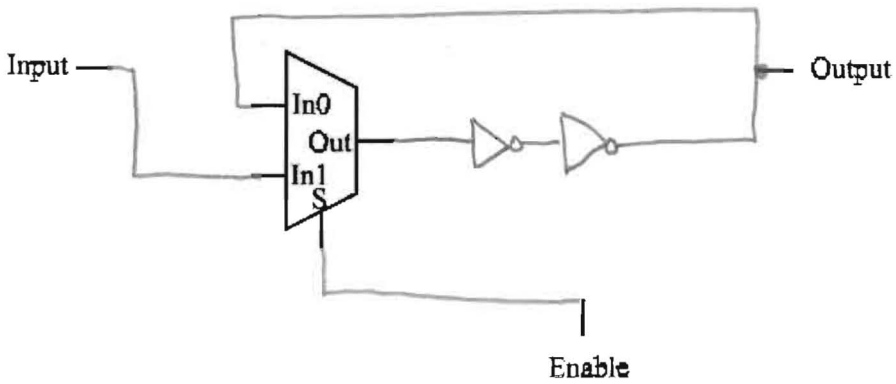


High Level Language		
Assembly Language		
Instruction Set		
Memory	Data Path	Controller
Storage	Functional Units	State Machines
Building Blocks		
Gate		
Switches and Wires		



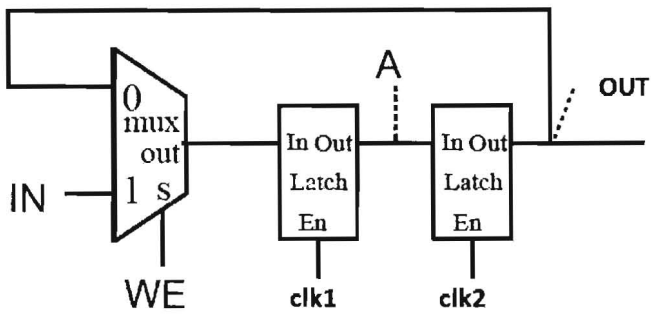
4) (5 points) Using a two-to-one mux, implement a transparent latch using basic gates (NAND, NOR, AND, OR, NOT, pass). Enable here is an active high write enable. (The block diagram of a transparent latch is show above just as an FYI only).

Try to minimize transistors.

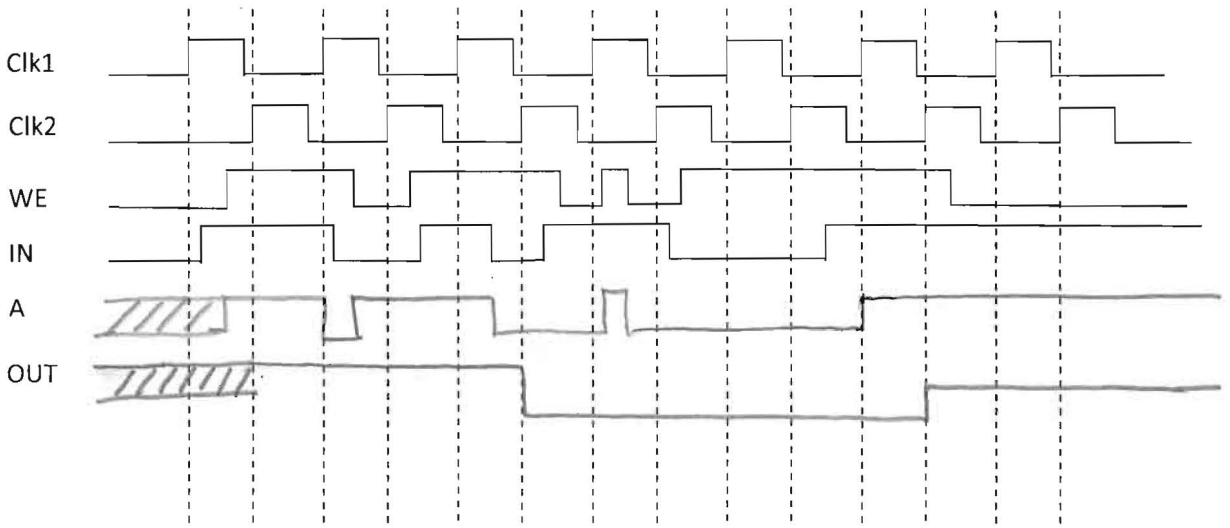


High Level Language		
Assembly Language		
Instruction Set		
Memory	Data Path	Controller
Storage	Functional Units	State Machines
Building Blocks		
Gate		
Switches and Wires		

5) (8 points) Consider the register implementation below.



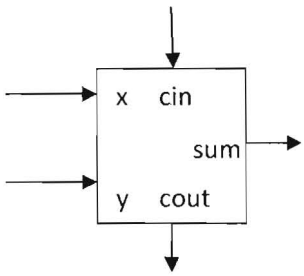
Assume the following signals are applied to the register above. Draw the signal at point A (output of the first latch), the signal at point OUT (output of second latch). Assume A and OUT start at logic unknown values.



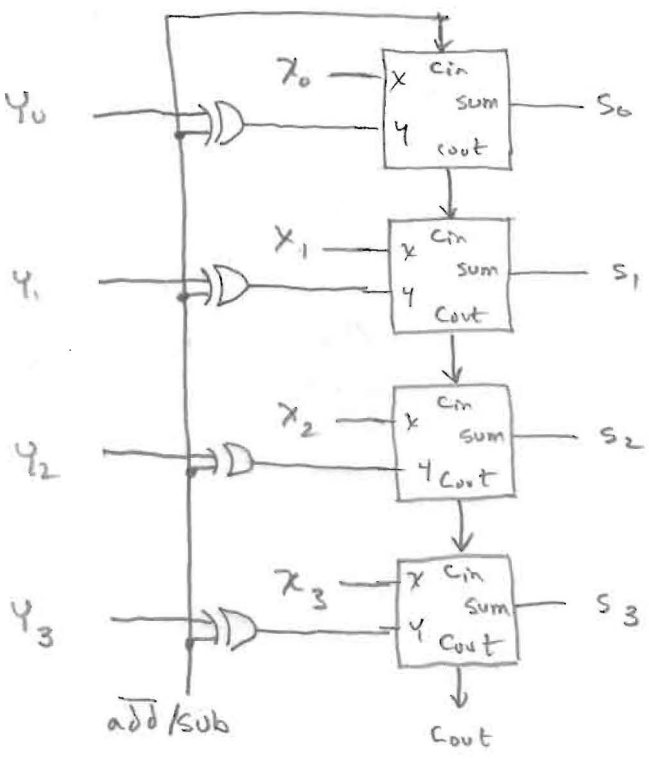
High Level Language		
Assembly Language		
Instruction Set		
Memory	Data Path	Controller
Storage	**Functional Units**	State Machines
Building Blocks		
Gate		
Switches and Wires		

6) a)(5 points) Complete the truth table below to the left for a "full adder" as shown below on the right.

X	Y	C_{in}	Sum	$Carry_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



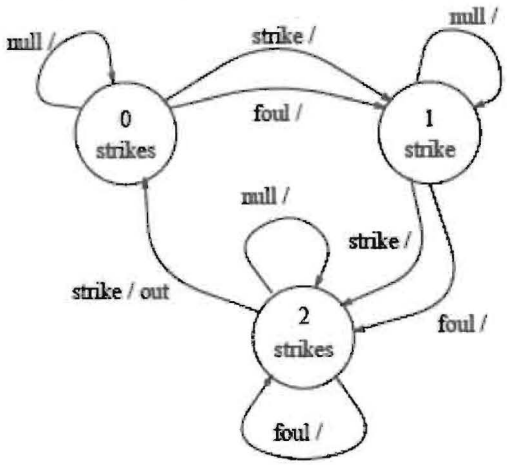
b) (10 Points) Using four of the full adder blocks shown above to the right, show how four of them would be used to implement a four bit adder that would be able to add/subtract two four bit numbers. Your inputs are X3, X2, X1, X0, Y3, Y2, Y1, Y0, $\overline{\text{add}}$/sub, S3, S2, S1, S0. You may use any additional logic gate types you want. Draw your hardware diagram below:



7) (10 points total) State machine problem straight from the practice problems:

Strike State Machine

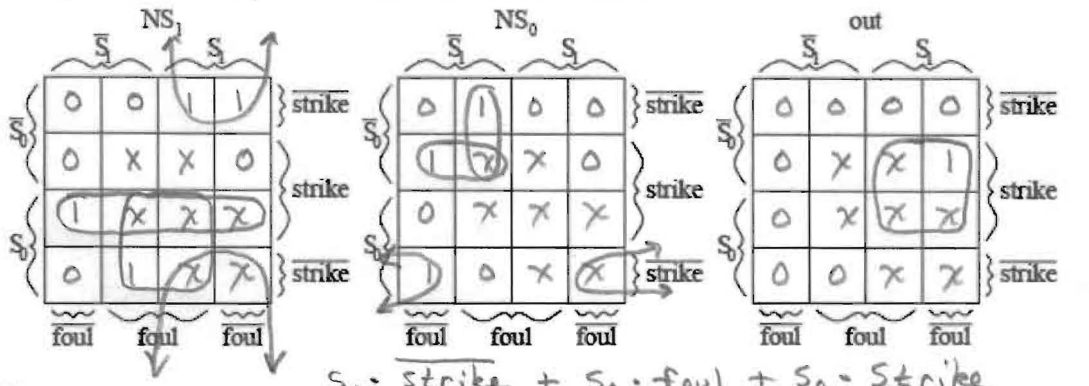
You're designing a baseball scoreboard. A sub-problem involves keeping track of strikes and fouls. A *strike* occurs when the batter swings at the ball and misses, or when the batter does not swing at a "good" pitch. If a batter earns three strikes during his/her turn at bat, he/she is "out". A *foul* occurs when the batter hits the ball, but the ball does not land in "fair" territory. Fouls are counted as strikes until the batter earns two strikes. Then fouls are ignored. The state diagram below captures these rules. The null input indicates neither a strike nor a foul occurred during that clock cycle. The strike and foul input cannot occur simultaneously, so the output for this case is don't care.



Part A Using the state diagram above, complete the state table for this diagram. The inputs are strike, foul, and state bits S_1 and S_0. The outputs are next state bits NS_1 and NS_0, and out.

S_1	S_0	*strike*	*foul*	NS_1	NS_0	*out*
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	1	0	0
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	0	1

Part B Determine the simplified logical expression for NS_1, NS_0, and *out* using your state table in part A. Complete the Karnaugh maps below, identify the prime implicants, and express the result.

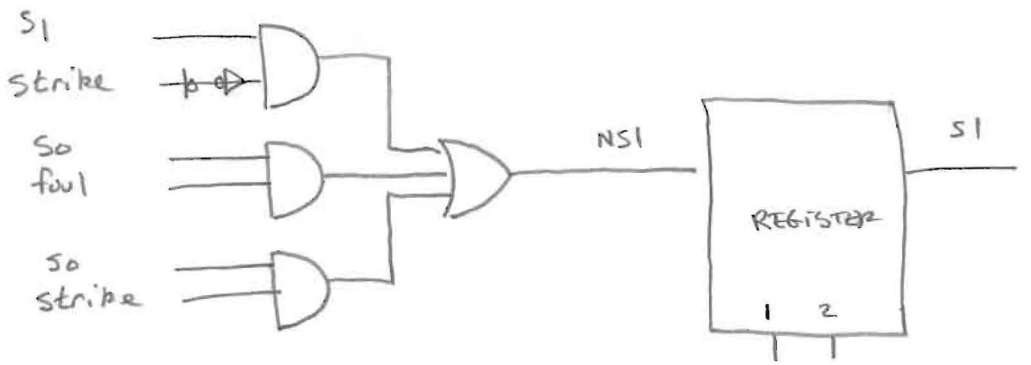


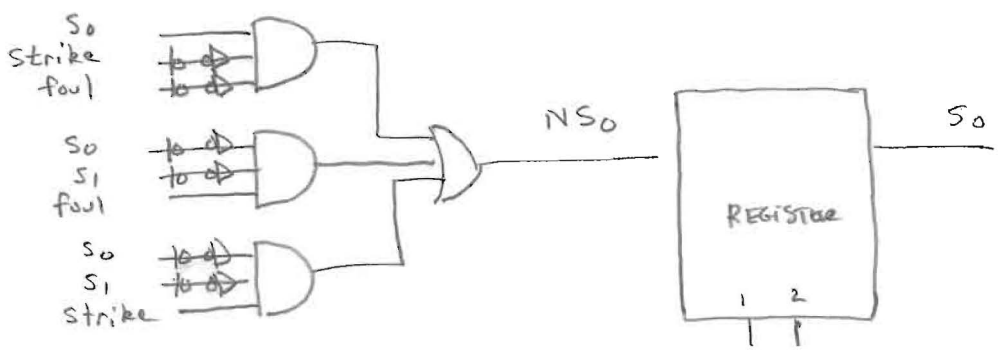
$NS_1 =$ $S_1 \cdot \overline{strike} + S_0 \cdot foul + S_0 \cdot Strike$

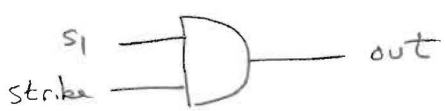
$NS_0 =$ $S_0 \cdot \overline{strike} \cdot \overline{foul} + \overline{S_1} \cdot \overline{S_0}\ foul + \overline{S_1} \cdot \overline{S_0} \cdot strike$

$out =$ $S_1 \cdot strike$

Part C Implement this state machine using your expression from part B, plus two register cells. Use an icon for a one bit register cell. You do not have to show the implementation of a register here. Your logic should be simplified.







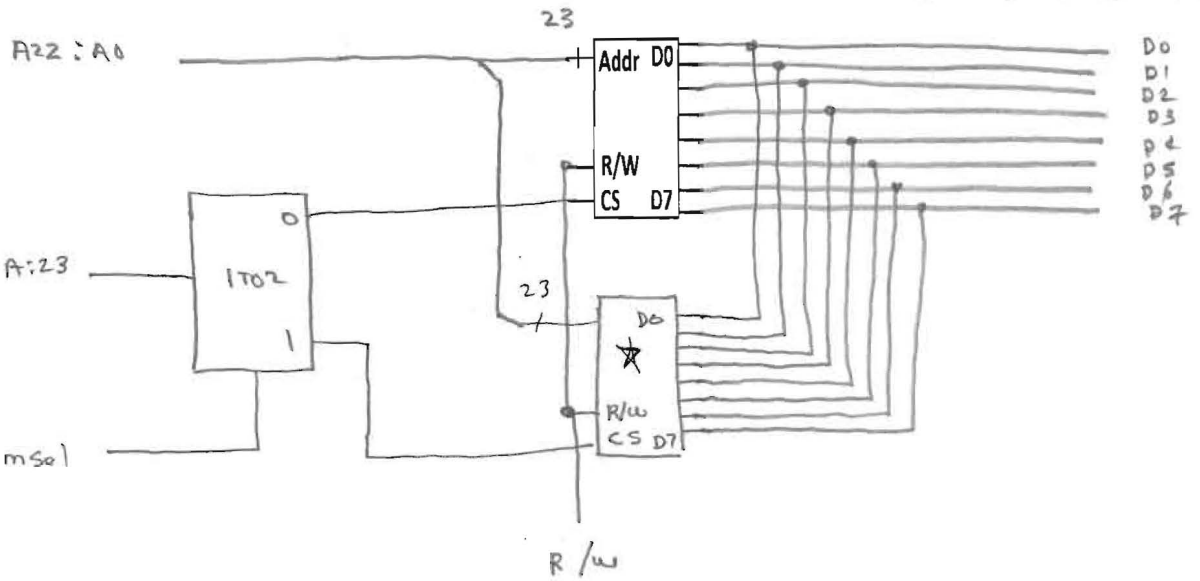
High Level Language		
Assembly Language		
Instruction Set		
Memory	Data Path	Controller
Storage	Functional Units	State Machines
Building Blocks		
Gate		
Switches and Wires		

8) (6 points) SIMM Memory System:

Using SIMMs that are 8 million addresses by 8 bit words:

Part A Suppose that these SIMMS are used to build a 16 million address memory system with 8 bit words. Using **however many** of these SIMMs you need, draw this memory system. In the drawing you make below, add as many SIMM chips as you need and then be sure you label the memory system inputs, Addr, R/W, and Mem Sel, and the system's outputs D0, D1, D2, etc. Also label bus widths, and inputs and outputs of any required decoders. **Put a star on the chip(s) containing the memory location addressed by an address containing a one followed all zeros (all of the address bits are input as a zero except the most significant bit which is a one).**

$8M \Rightarrow 8 \cdot 2^{20} = 2^3 2^{20} = 2^{23}$



High Level Language		
Assembly Language		
Instruction Set		
Memory	Data Path	Controller
Storage	Functional Units	State Machines
Building Blocks		
Gate		
Switches and Wires		

9) Using the attached single cycle datapath we have discussed extensively in class (see last page of exam for part of it), implement the microcode for the following instructions (see instruction examples also on last page):

a) (5 points) addi $3, $2, 25

#	X	Y	Z	rwe	imen	im va	auen	-a/s	luen	lf	suen	st	lden	sten	r/-w	msel	description
1	2	X	3	1	1	25	1	0	0	XXXX	0	XX	0	0	X	0	$3=$2+25

b) (5 points) sw $1, ($2)

#	X	Y	Z	rwe	imen	im va	auen	-a/s	luen	lf	suen	st	lden	sten	r/-w	msel	description
1	2	1	X	0	0	X	0	X	0	XXXX	0	XX	0	1	0	1	memory[$2]=$1

c) (5 points) Thinking about the "putting it all together" system block diagram: Which bit fields in the part (a) microcode you wrote above would be stored in a Read Only Memory controller? For those fields not stored in the Read Only memory controller where would they come from?

THE REGISTER SELECTION SIGNALS FOR X, Y, Z AND THE IMMEDIATE VALUE OF 25 ARE CONTAINED IN THE ACTUAL INSTRUCTION STORED IN MEMORY

ALL OTHER SIGNALS (rwe, imm en, etc) ARE STORED IN READ ONLY MEMORY CONTROLLER

High Level Language		
Assembly Language		
Instruction Set		
Memory	Data Path	Controller
Storage	Functional Units	State Machines
Building Blocks		
Gate		
Switches and Wires		

10) (5 points) Instruction Format. For the third instruction type we discussed, the Immediate Instruction Format, show the actual 32 bits for the instruction:

addi $10, $8, 4

assuming the opcode for addi is 001000.

OPCODE	Z	X	IMMEDIATE VALUE 4
0 0 1 0 0 0	0 1 0 1 0	0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

9) Assembly language programming

Consider the following MIPS program fragment. The attached last exam page lists the instruction set and explains them with examples.

address	label	instruction
1000	start:	addi $1,$0,300
1004		lw $2,($1)
1008	loop:	addi $1,$1,4
1012		lw $4,($1)
1016		slt $3,$2,$4
1020		beq $3,$0,skip1
1024		add $2,$0,$4
1028	skip1:	slti $3,$1,324
1032		bne $3,$0,loop
1036		add $5,$0,$2

Assume memory holds the following starting at address 300:

Address	Data
300	-16
304	-7
308	-12
312	5
316	8
320	-3
324	10
328	-19
332	34
336	20

Part a) (5 points) How many words of data are read in by the program fragment?

Total number of data words read: 7 300 → 324 INCLUSIVE

Part b) (5 points) How many times is the add instruction at address 1024 executed?
Number of times: 4

Part c) (5 points) What are the values in the following registers at the end of the program fragment?
$1 = 324
$2 = 10
$3 = 0
$4 = 10
$5 = 10

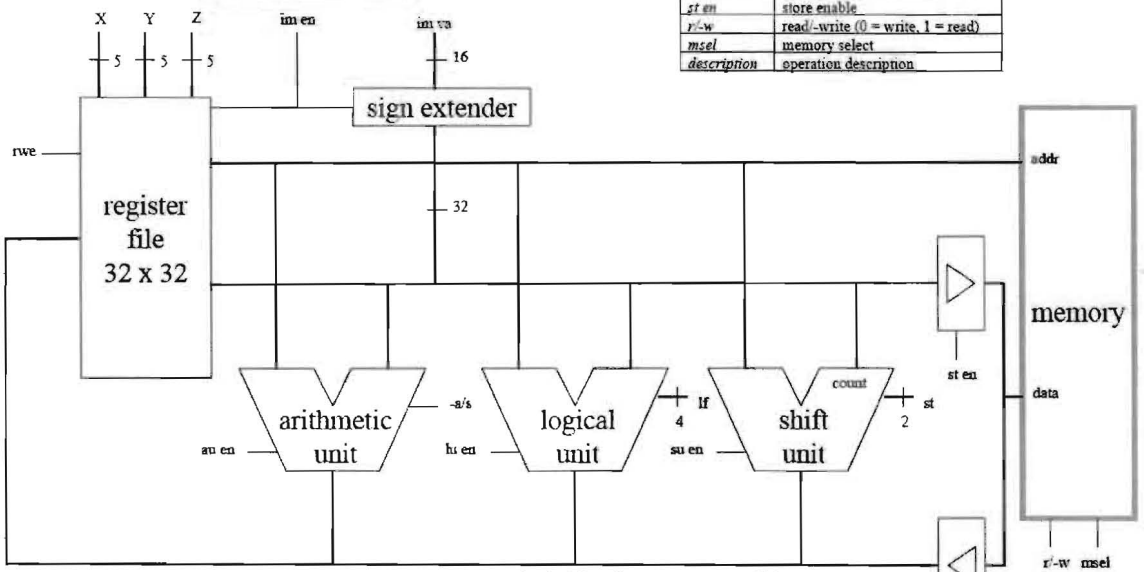
Part d) (5 points) What does this program fragment do?

FINDS MAX VALUE IN MEMORY 300 TO 324 INCLUSIVE

You may tear this off from the exam and you do not need to turn in this page.

cycle	cycle number
X	register driven onto X bus
Y	register driven onto Y bus
Z	register written from Z bus
rwe	register write enable
im en	immediate enable on Y bus
im va	immediate value

au en	arithmetic unit enable
-a/s	-add / sub (0 = add, 1 = subtract)
lu en	logical unit enable
lf	logical function
su en	shift unit enable
st	shift type
ld en	load enable
st en	store enable
r/-w	read/-write (0 = write, 1 = read)
msel	memory select
description	operation description



logical functions		
X	Y	out
0	0	lf_0
1	0	lf_1
0	1	lf_2
1	1	lf_3

shift types
0 = logical
1 = arithmetic
2 = rotate
+ count shifts right
- count shifts left

instruction	example	meaning
add	`add $1,$2,$3`	`$1 = $2 + $3`
subtract	`sub $1,$2,$3`	`$1 = $2 - $3`
add immediate	`addi $1,$2,100`	`$1 = $2 + 100`
multiply	`mul $1,$2,$3`	`$1 = $2 * $3`
divide	`div $1,$2,$3`	`$1 = $2 / $3`
and	`and $1,$2,$3`	`$1 = $2 & $3`
or	`or $1,$2,$3`	`$1 = $2 | $3`
xor	`xor $1,$2,$3`	`$1 = $2 xor $3`
and immediate	`andi $1,$2,100`	`$1 = $2 & 100`
or immediate	`ori $1,$2,100`	`$1 = $2 | 100`
xor immediate	`xori $1,$2,100`	`$1 = $2 xor 100`
shift left logical	`sll $1,$2,5`	`$1 = $2 << 5 (logical)`
shift right logical	`srl $1,$2,5`	`$1 = $2 >> 5 (logical)`
shift left arithmetic	`sla $1,$2,5`	`$1 = $2 << 5 (arithmetic)`
shift right arithmetic	`sra $1,$2,5`	`$1 = $2 >> 5 (arithmetic)`
load word	`lw $1, ($2)`	`$1 = memory [$2]`
store word	`sw $1, ($2)`	`memory [$2] = $1`
load upper immediate	`lui $1,100`	`$1 = 100 x` 2^{16}
branch if equal	`beq $1,$2,100`	`if ($1 = $2), PC = PC + 4 + (100*4)`
branch if not equal	`bne $1,$2,100`	`if ($1 ≠ $2), PC = PC + 4 + (100*4)`
set if less than	`slt $1, $2, $3`	`if ($2 < $3), $1 = 1 else $1 = 0`
set if less than immediate	`slti $1, $2, 100`	`if ($2 < 100), $1 = 1 else $1 = 0`
jump	`j 10000`	`PC = 10000*4`
jump register	`jr $31`	`PC = $31`
jump and link	`jal 10000`	`$31 = PC + 4; PC = 10000*4`