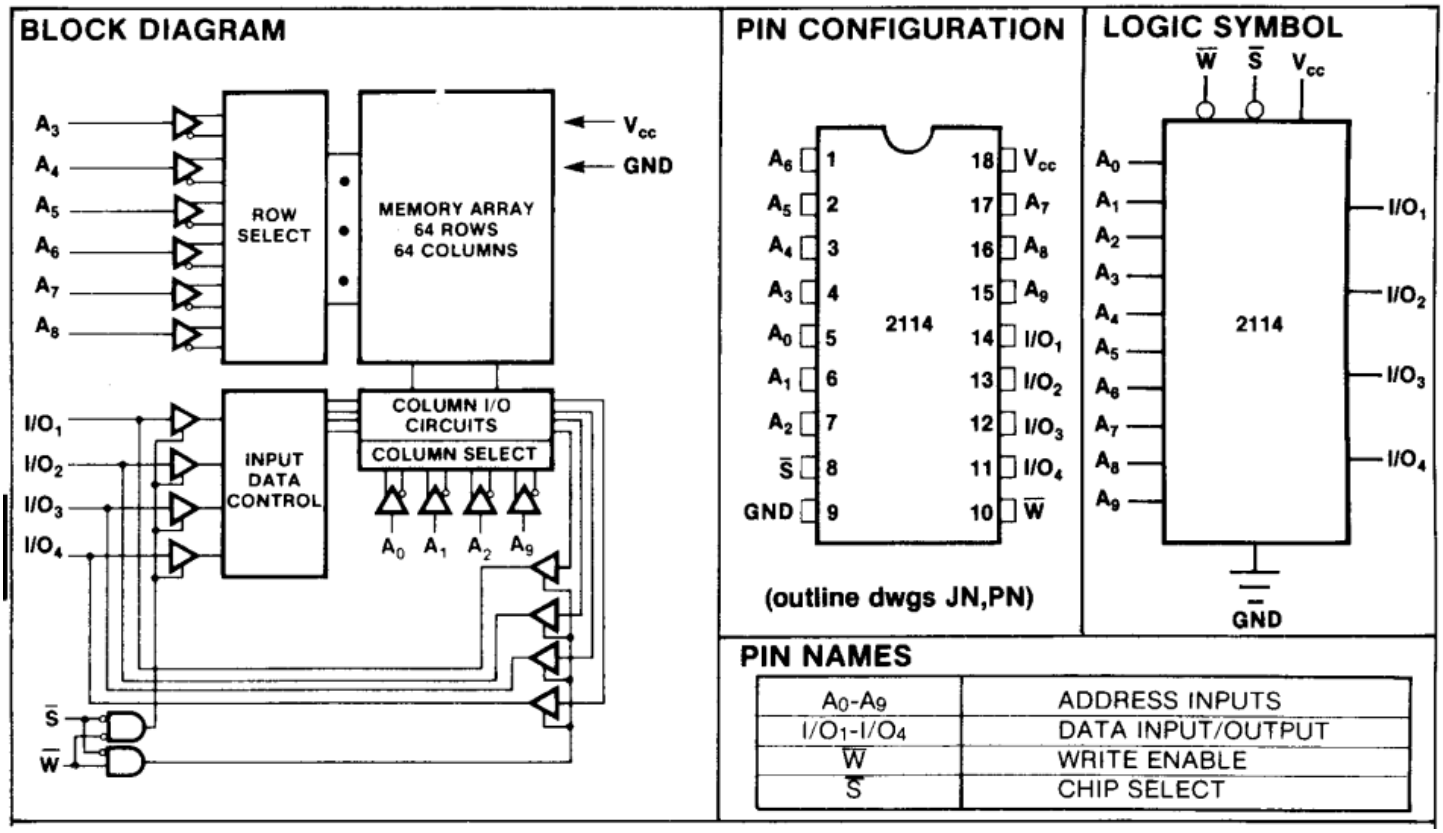
Problem 1 (25 points)

Here is some information about the 2114 RAM chip, which was used for video color RAM in the Commodore 64.



PIN NAMES

A_0-A_9	ADDRESS INPUTS
I/O_1-I/O_4	DATA INPUT/OUTPUT
$\overline{W}$	WRITE ENABLE
$\overline{S}$	CHIP SELECT

a) How many words of memory does this chip contain?

$2^{10} = 1k = 1024$

b) How many bits are in each word?

4

c) In the block diagram, the memory array is described as 64 rows and 64 columns. Is that referring to the words or to the bit cells?

Bit cells

d) How would the memory array be described if it was the opposite of your answer from #3? I.e. if you said it was referring to the bit cells, what would the description be if it was referring to the words, and vice versa.

64 rows x 16 columns

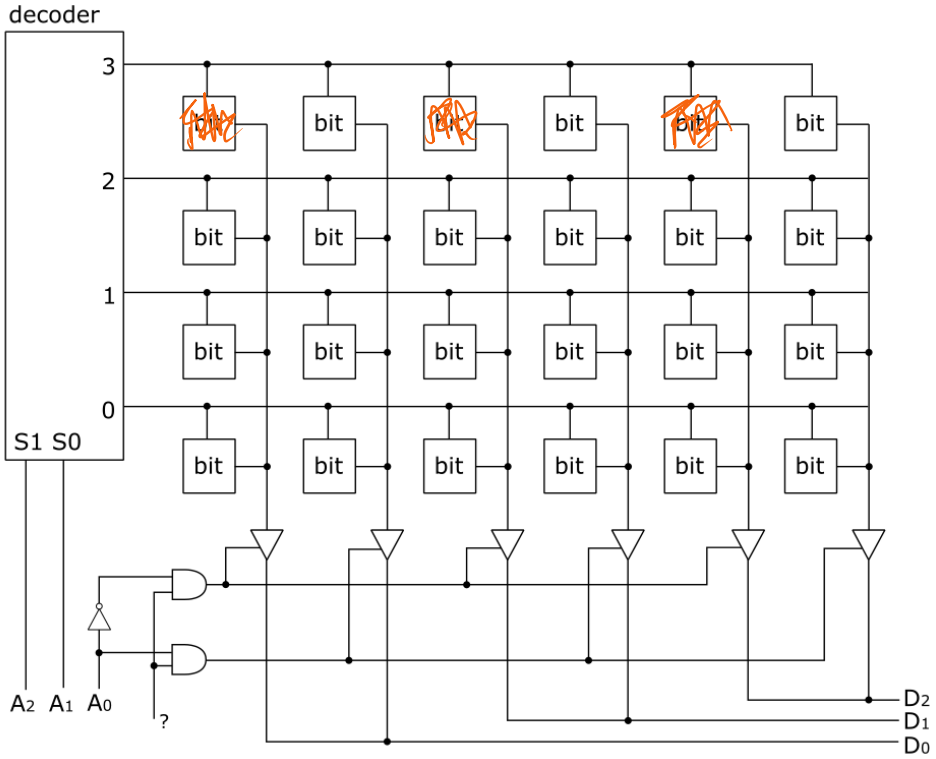
e) If a different chip contained four times as many words, how many <u>additional</u> address pins would be needed?

Two more

f) Based on the block diagram, what combination(s) of $\overline{S}$ and $\overline{W}$ will tri-state the I/O pins? Assume that the tri-state drivers are active when their control signal is 1.

00, 10, 11

Problem 2 (10 points)



a) Based on the memory devices we've seen in this class, what is an appropriate name for the signal marked with a question mark in the bottom left of the diagram?

chip select, chip enable, output enable, etc.

b) Color in the cells in the diagram above that contain the data at address 6.

The datapath reference is on the last page. You can tear it off if you want.

Problem 3 (15 points)

In the MIPS datapath, all of the following effectively implement `R2 = R1`:

- `R2 = R1 OR 0`
- `R2 = R1 OR R0`
- `R2 = R1 + 0`
- `R2 = R1 + R0`
- Setting st_en and ld_en to 1, disabling the arithmetic, logic, and shift units and memory, and using the Y bus to transfer R1 to R2.

Describe two other unique ways to accomplish `R2 = R1` in the MIPS datapath. You can use existing instructions, or you can describe what the datapath components should do or what the control signals should be.

1) shift by 0
AND with FFFFFFFF

2) XOR with 0
set LF to identity

The datapath diagram is on the last page. You can tear it off if you want.

Problem 4 (25 points)

4a) Use the description column to fill in the datapath signals to implement that operation.

X	Y	Z	rwe	im en	im_va	au en	-a /s	lu en	lf_{3-0}	su en	st	st en	ld en	-r /w	msel	description
3	7		0	0		0		0		0		1	0	1	1	MEM[R3] = R7
2		7	1	1	-3	0		0		1	0	0	0	0	0	R7 = R2 SRL 3

4b) From the provided microcode, write a description of the operation, either as an assembly code instruction or as a plain-English description. Simplify the description of the operation if possible – for example, like in problem 3, if the datapath performs `R2 = R1 OR 0`, it would be better to write `R2 = R1` or "R2 gets the value of R1". Note that everything not bolded is the same in all rows. Remember that the immediate is sign-extended.

X	Y	Z	rwe	im en	im_va	au en	-a /s	lu en	Lf_{3-0}	su en	st	st en	ld en	-r /w	msel	description
3	2	1	1	**1**	**$FFFF_{16}$**	**0**	0	**1**	**1110**	0	0	0	0	0	0	Set R1 to all 1s
3	2	1	1	**1**	**0**	**0**	0	**1**	**1111**	0	0	0	0	0	0	Set R1 to all 1s
3	2	1	1	**1**	**0**	**1**	0	**0**	**0000**	0	0	0	0	0	0	R1 = R3
3	2	1	1	**0**	**$FFFF_{16}$**	**1**	0	**0**	**0000**	0	0	0	0	0	0	R1 = R3 + R2

Problem 5 (25 points)

- Assume that R1 contains a value between 0 and 31, and consider that the "input" to the code.
- The "output" is in R5 after the code completes.

```
R2 = 32              #
R2 = R2 - R1         #
R5 = -1              #
R5 = R5 SRL R2       # shift by amount stored in R2
```

a) Describe the result of the code in terms of the input and output. Hint: the operation is mathematical in nature, although it can be also described in terms of the bits if you don't see the mathematical connection.

Puts $2^{R1}-1$ in R5

E.g. if R1 = 4, R3 = 0...01111

b) The author of the code forgot that the shift amount for the MIPS shifter is 5 bits (refer to the datapath reference). It comes from the least-significant 5 bits of the Y bus. How does that affect the behavior of the code for the intended input range of 0-31?

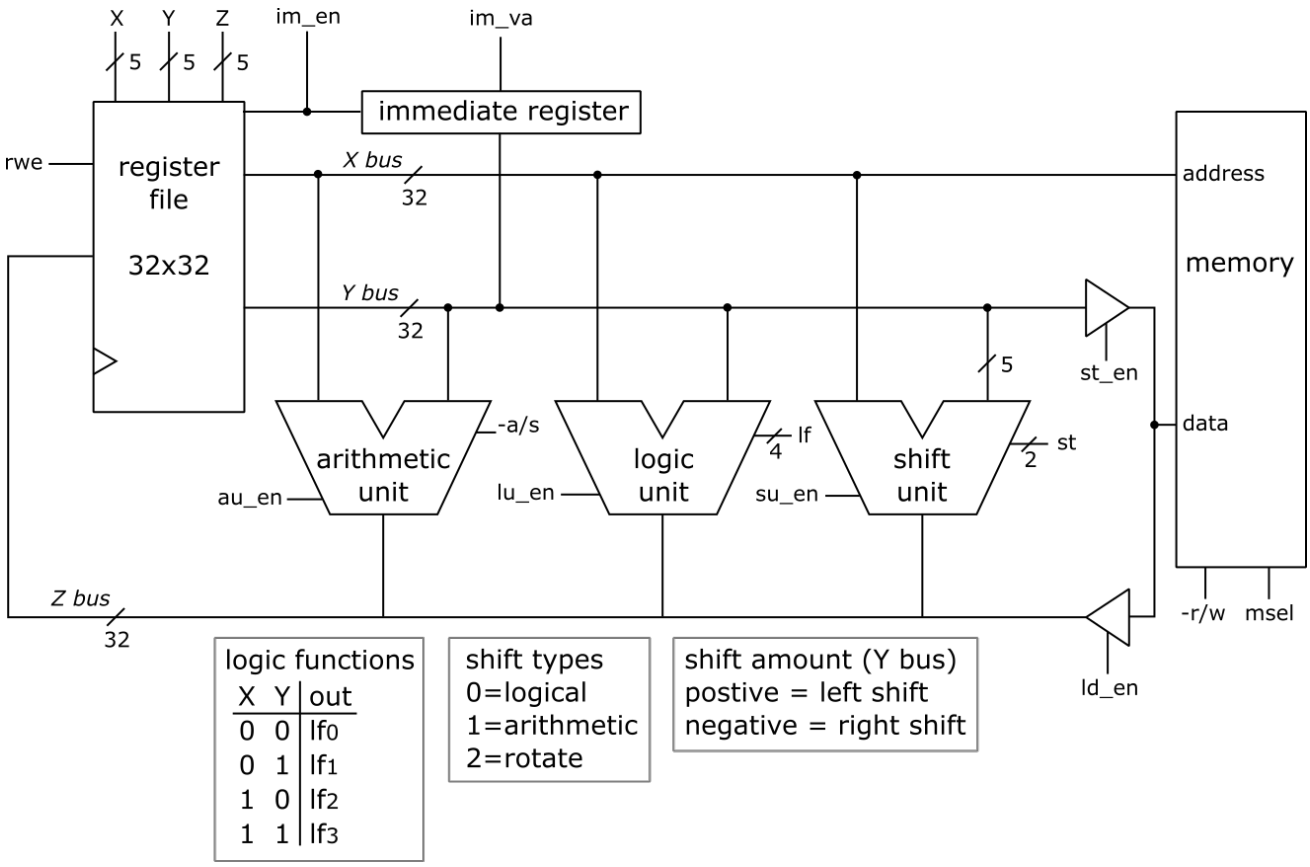
(note that you can ignore any related incorrect behavior in your answer to part a)

Input numbers 0-16 don't work, because shifting 16+ places would shift the wrong direction, because the most-significant bit of shift amount is the direction.

Bonus (1 possible bonus point)

If you've finished early, fix the code so that it works for all input values 0-31. You can write actual code, or you can describe the fix as long as it's clear how it would be translated to assembly code.

split the shift into multiple shifts when needed.



Signal	Description	Signal	Description
X, Y	Read register addresses	lf	Logic function (see table below diagram))
Z	Write register address	su_en	Shift unit enable
rwe	Register write enable	st	Shift type (see table below diagram)
im_en	Immediate enable	st_en	Store enable
im_va	Immediate value	ld_en	Load enable
au_en	Arithmetic unit enable	-r/w	Read/write memory (0=read, 1=write)
-a/s	Add/subtract (0=add, 1=subtract)	msel	Memory select (memory output enable)
lu_en	Logic unit enable		

instruction	RTL description
add	$d = $s + $t;
subtract	$d = $s - $t;
add immediate	$t = $s + imm;
and	$d = $s AND $t;
or	$d = $s OR $t;
xor	$d = $s XOR $t;
and immediate	$t = $s AND imm;
or immediate	$t = $s OR imm;
xor immediate	$t = $s XOR imm;
shift left logical	$d = $t SLL a;
shift right logical	$d = $t SRL a;
shift left arithmetic	$d = $t SLA a;
shift right arithmetic	$d = $t SRA a;
load word	$t = MEM[$s];
store word	MEM[$s] = $t;
load immediate	$t = imm;
branch if equal	IF $s = $t GOTO label;
branch if not equal	IF $s != $t GOTO label;
set if less than	$d = $s < $t; (if $s < $t $d = 1; else $d = 0;)
set if less than immediate	$d = $s < imm; (if $s < imm $t = 1; else $t = 0;)
jump	GOTO label;