Question 1) (20 points)

Shift these values as described, using 6 bits for each result.

1a)
`000100 logical left-shift by 2:` 010000

1b)
`101011 logical right-shift by 2:` 001010

1c)
`000101 arithmetic left-shift by 2:` 010100

1d)
`101011 arithmetic right-shift by 2:` 111010

1e)
`101011 barrel right-shift by 2:` 111010

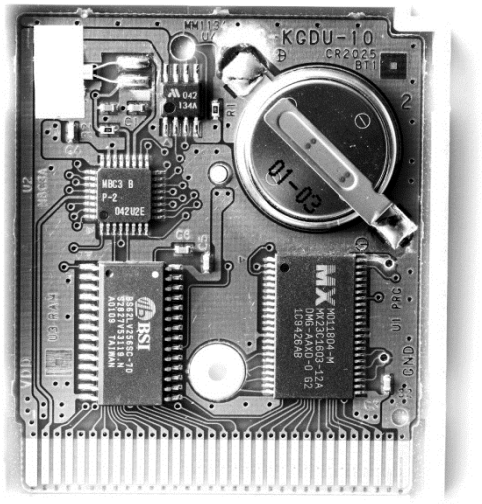
Question 2) (20 points)

Fill in the requested microcode signals needed to make the MIPS datapath implement the given instruction (just those two signals for each instruction). If the value doesn't matter, write 'x'. The datapath reference is on the last page, and can be removed.

To execute:	Microcode signals:	
R6 = R7 + R8	x = 7	rwe = 1
R3 = R2 XOR 1	y = X	su_en = 0
R7 = MEM[R9]	z = 7	st_en = 0
MEM[R7] = R9	-r/w = 1	rwe = 0

Question 3) (25 points)

This is a Pokémon Silver cartridge for the Game Boy Color.
The memory chip in the lower-right of the cartridge stores the program. It's a 2M-word memory, which means it needs 21 address signals. But there's a problem: the cartridge only has 16 address pins.



To allow for more memory than the address bus can address at once, a chip inside the cartridge called the "memory bank controller" provides some of the address bits. The Game Boy can only "see" 16k words of program memory at a time, but by making the memory bank controller change its outputs, a different set of 16k words can be accessed. These swappable chunks of memory are called "banks".
[The diagram below might help you work through these questions]

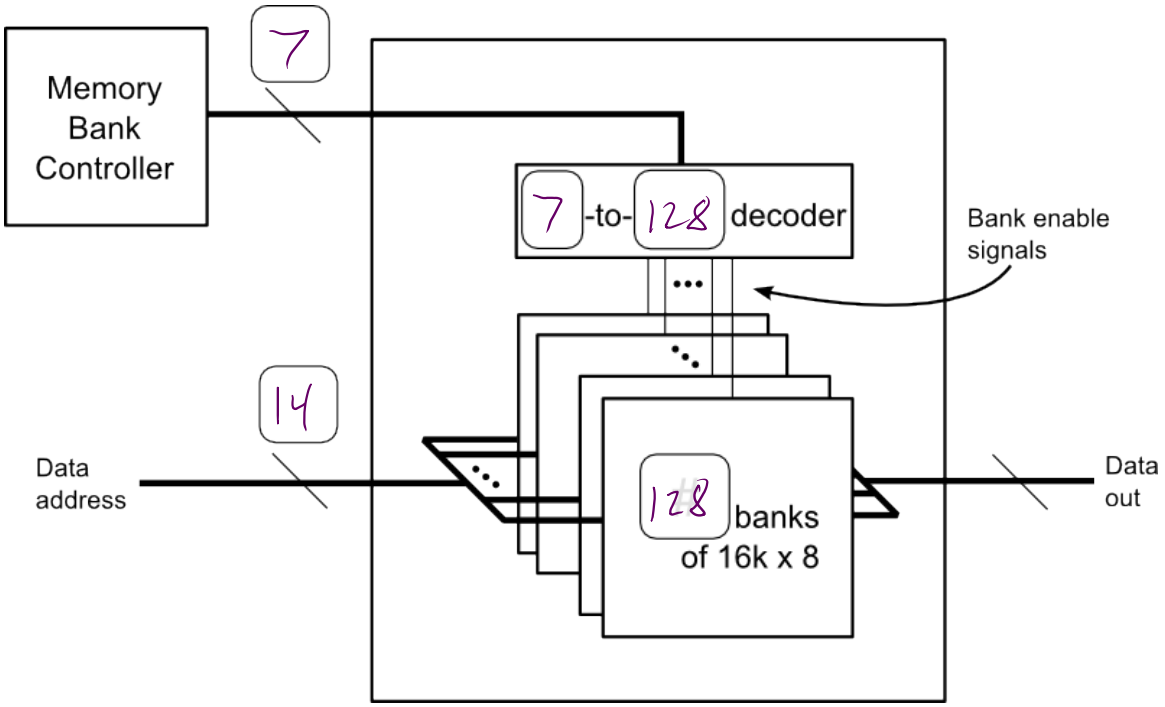
3a) Since each bank is 16k words, how many bits are required to address each word inside a bank? 14 bits $(2^4 \cdot 2^{10}$ words$)$

3b) If the memory chip is 2M words and it's broken up into banks of 16k words, how many banks are there?

128 banks (express your answer with k or M if needed) $2M/16k = 2^{21}/2^{14} = 2^7$

3c) From your answer to 3b, how many address bits are needed to select the bank? 7 bits

3d) Based on all that, fill in the blanks on this diagram (the rounded squares), describing the effective memory layout. Blanks on buses are asking for bus width.



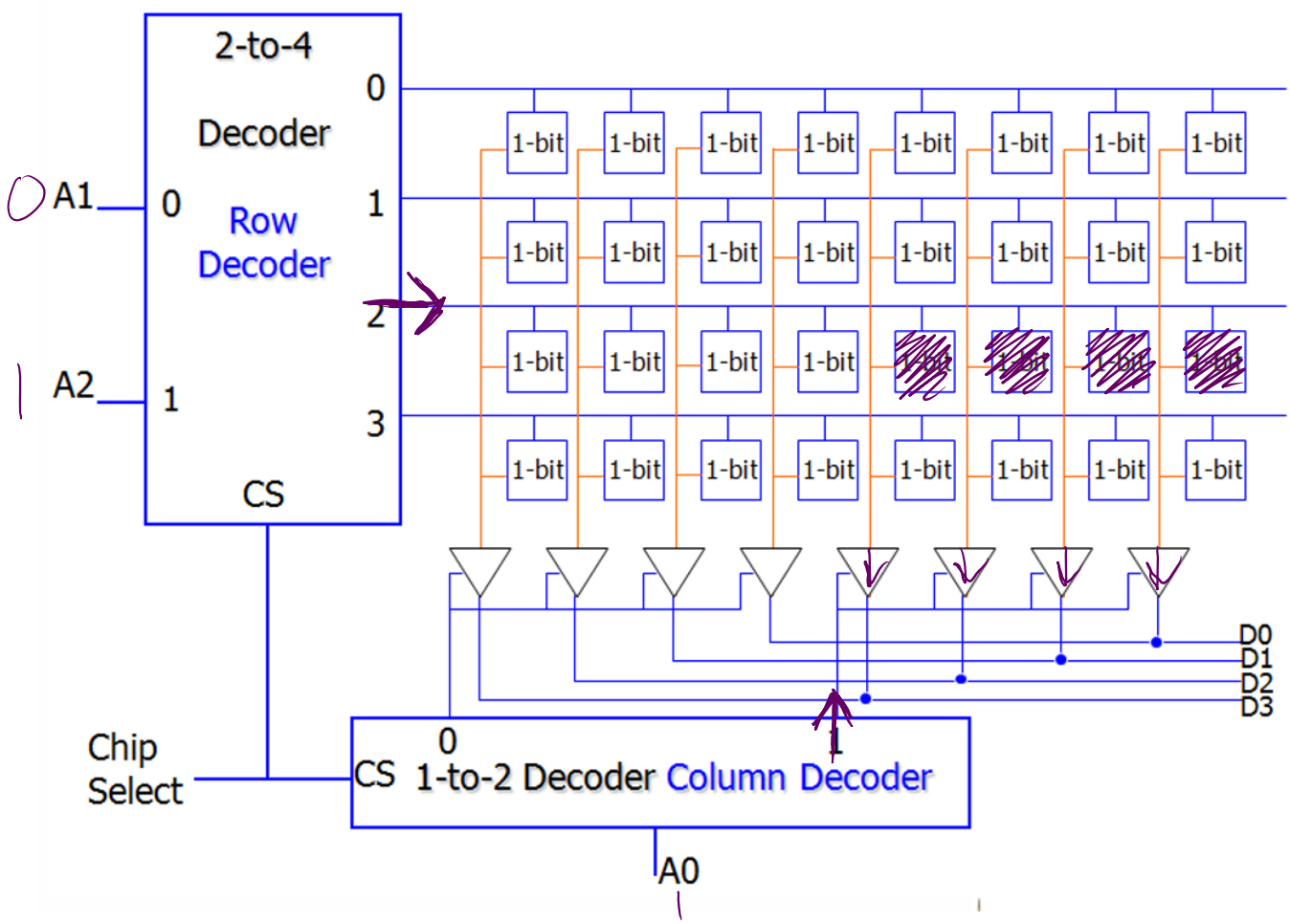
3e) The memory bank controller in Game Boy Color games could supply *up to* 9 bits of address. What is the maximum program size for Game Boy Color games?

8M words (express in terms of k, M, etc.) $2^{14} \cdot 2^{9} = 2^{23} = 8M$

Question 4) (10 points)

Color in the bit cells in this diagram that contain the data at address 5.

A = 101



Question 5) (25 points)

BCD (binary coded decimal) is a number format where each digit of the decimal representation of a number gets encoded into four bits. For example, since 13_{10} is "1" followed by "3", it would be encoded in BCD as 00010011 (four bits representing "1" followed by four bits representing "3").

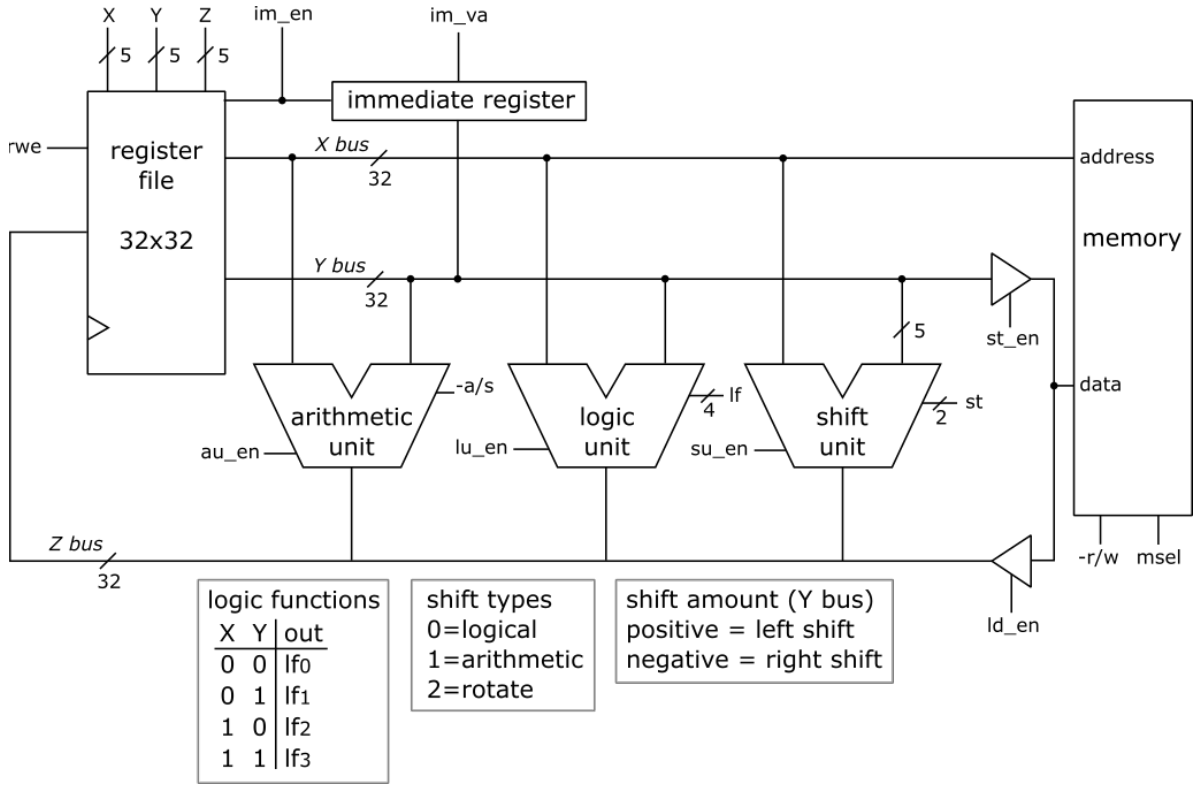
Write a snippet of assembly code (in "RTL" format) that converts a number between 0 and 19 into BCD. Assume that the number is in register R1 at the beginning of the snippet, and leave the result in R1.

For example, 00000000000000000000000000010010 (18_{10}) should become 00000000000000000000000000011000 (four bits representing "1" followed by four bits representing "8").

Keep in mind that it only needs to work for input values 0-19.

Many ways to do this. Here's one way.

	MIPS ASM for reference
; check if # is ≥ 10	
R3 = R1 < 10	slti $3, $1, 10
IF R3 != R0 Goto Done	bne $3, $0, Done
; need to add 6 to get to BCD	
R1 = R1 + 6	addi $1, $1, 6
Done:	Done:
; R1 is now in BCD	



Signal	Description	Signal	Description
X, Y	Read register addresses	lf	Logic function (see below)
Z	Write register address	su_en	Shift unit enable
rwe	Register write enable	st	Shift type (see below)
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
lu_en	Logic unit enable		

instruction	assembly	RTL description
add	`add $d, $s, $t`	$d = $s + $t;
subtract	`sub $d, $s, $t`	$d = $s - $t;
add immediate	`addi $t, $s, imm`	$t = $s + imm;
and	`and $d, $s, $t`	$d = $s AND $t;
or	`or $d, $s, $t`	$d = $s OR $t;
xor	`xor $d, $s, $t`	$d = $s XOR $t;
and immediate	`andi $t, $s, imm`	$t = $s AND imm;
or immediate	`ori $t, $s, imm`	$t = $s OR imm;
xor immediate	`xori $t, $s, imm`	$t = $s XOR imm;
shift left logical	`sll $d, $t, a`	$d = $t SLL a;
shift right logical	`srl $d, $t, a`	$d = $t SRL a;
shift left arithmetic	`sla $d, $t, a`	$d = $t SLA a;
shift right arithmetic	`sra $d, $t, a`	$d = $t SRA a;
load word	`lw $t, ($s)`	$t = MEM[$s];
store word	`sw $t, ($s)`	MEM[$s] = $t;
load upper immediate	`lui $t, imm`	$t = imm SLL 16;
load immediate	`li $t, imm / addi $t, $0, imm`	$t = imm;
branch if equal	`beq $s, $t, offset`	if $s==$t GOTO [label];
branch if not equal	`bne $s, $t, offset`	if $s!=$t GOTO [label];
set if less than	`slt $d, $s, $t`	$d = $s < $t; ($d = 1 if true, else $d = 0)
set if less than immediate	`slti $t, $s, imm`	$d = $s < imm; ($d = 1 if true, else $d = 0)
jump	`j target`	GOTO [label];
jump register	`jr $s`	PC = $s;

More lined paper if you need it, but make sure I know that there's something here if I need to grade it.