Problem 1 (2 parts, 18 points) Datapath Elements

Part A (9 points) Consider the following input and output values for a shift operation. Determine the shift *type* and *amount* required to achieve the listed transformation. I/Os are in hexadecimal.

Input Value	Output Value	Shift Type	Shift Amount (signed decimal value)
87654321	FFF87654	arithmetic	+12
87654321	00000008	logical	+28
87654321	43218765	rotate	16

Part B (9 points) Consider the following input and output values for a logical operation. Determine the *logical function* and *function code* (in hexadecimal) required for the operation.

X Input	Y Input	Output	Logical Function	Function Code
87654321	00FF00FF	00650021	AND	8
87654321	00FF00FF	879A43DE	XOR	6
87654321	00FF00FF	789ABCDE	$\overline{X}$	5

Problem 2 (3 parts, 32 points) Memory Systems

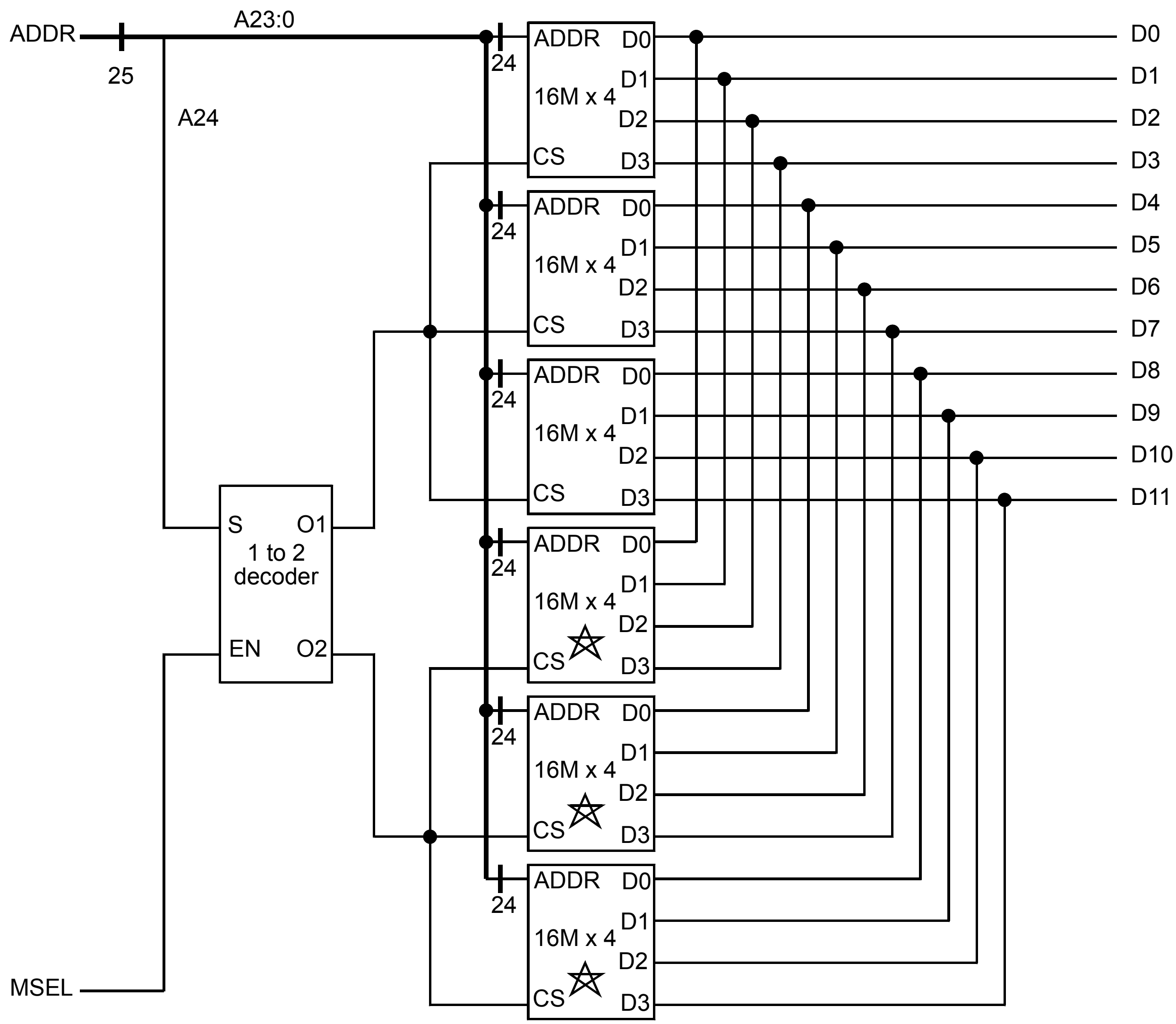
Part A (12 points) Consider a DRAM chip organized as **64 million addresses** of **16 bit words**. Assume both the DRAM cell and the DRAM chip is square. The column number and offset concatenate to form the memory address. Using the organization approach discussed in class, answer the following questions about the chip. **Express all answers in decimal**.

number of columns	64Mx16=1G; sqrt(2^30) = 2^15 = 32K
column decoder required (*n* to *m*)	15 to 32K
type of mux required (*n* to *m*)	32K/16 = 2K to 1
number of muxes required	16
number of address lines in column number	15
number of address lines in column offset	11

Part B (10 points) Consider a **4 Gbyte** memory system with **512 million addresses** of **8 byte words** using DRAM chips organized as **64 million** addresses by **16 bit words.**

word address lines for memory system	$\log_2(512M) = 29$
chips needed in one bank	8 bytes * 8 bits/byte / 16 bits = 4
banks for memory system	512M / 64M = 8
memory decoder required (*n* to *m*)	3 to 8
DRAM chips required	4 chips / bank x 8 banks = 32 chips

Part C (10 points) Design a 32 M address x 12 bit memory system with six 16 M address x 4 bit memory chips. **Label all busses and indicate bit width**. Assume R/W is connected and not shown here. Use a decoder if necessary. Place a star on the chip(s) that contain address 26,000,000.



Problem 3 (5 parts, 25 points) Microcode

Using the supplied datapath, write microcode fragments to accomplish the following procedures. Express all values in hexadecimal notation. Use 'X' when a value is don't cared. Smile when you're happy. For maximum credit, complete the description field.

Part A (6 points) $6 ← mem[0x3000]. **Use only register 6.**

#	X	Y	Z	rwe	im en	im va	au en	-a /s	lu en	lf	su en	st	ld en	st en	r/ -w	msel	description
1	X	X	6	1	1	3000	0	X	1	C	0	X	0	0	X	0	$6 ← 0x3000
2	6	X	6	1	0	X	0	X	0	X	0	X	1	0	1	1	$6 ← mem[$6]

Part B (5 points) Mask all but the eight least significant bits of $8 **Use only register 8.**

#	X	Y	Z	rwe	im en	im va	au en	-a /s	lu en	lf	su en	st	ld en	st en	r/ -w	msel	description
1	8	X	8	1	1	FF	0	X	1	8	0	X	0	0	X	0	$8 ← $8 & 0xFF

Part C (4 points) Subtract 8 from register $3. **Use only registers 3.**

#	X	Y	Z	rwe	im en	im va	au en	-a /s	lu en	lf	su en	st	ld en	st en	r/ -w	msel	description
1	3	X	3	1	1	8	1	1	0	X	0	X	0	0	X	0	$3 ← $3 - 8

Part D (6 points) mem[$4] ← 0x1957. **Use only registers 4 and 5.**

#	X	Y	Z	rwe	im en	im va	au en	-a /s	lu en	lf	su en	st	ld en	st en	r/ -w	msel	description
1	X	X	5	1	1	1957	0	X	1	C	0	X	0	0	X	0	$5 ← 0x1957
2	4	5	X	0	0	X	0	X	0	X	0	X	0	1	0	1	mem[$4] ← $5

Part E (4 points) $7 ← 32 * $7. **Use only register 7.**

#	X	Y	Z	rwe	im en	im va	au en	-a /s	lu en	lf	su en	st	ld en	st en	r/ -w	msel	description
1	7	X	7	1	1	3B	0	X	0	X	1	1	0	0	X	0	$7 ← $7 << 5

Problem 4 (1 part, 25 points) Assembly Programming

Part A (25 points) Complete this subroutine that averages an array of integers in memory. Assume $4 contains the number of integers and $5 contains the starting address of the array. Return the average in $7. Use the $8 for the input value, $7 for the sum, and $6 for the end test.

label	*instruction*	*comment*
`AvgN:`	`sll $6, $4, 2`	`# offset = 4 * # elems`
	`add $6, $6, $5`	`# end = base + offset`
	`lw $7, 0($5)`	`# load 1st element`
	`addi $5, $5, 4`	`# point to 2nd element`
`Loop:`	`lw $8, 0($5)`	`# load next element`
	`add $7, $7, $8`	`# add to sum`
	`addi $5, $5, 4`	`# point to next element`
	`bne $5, $6, Loop`	`# loop until end`
	`div $7, $4`	`# compute average`
	`mflo $7`	`# put average in $7`
	`jr $31`	`# return to caller`