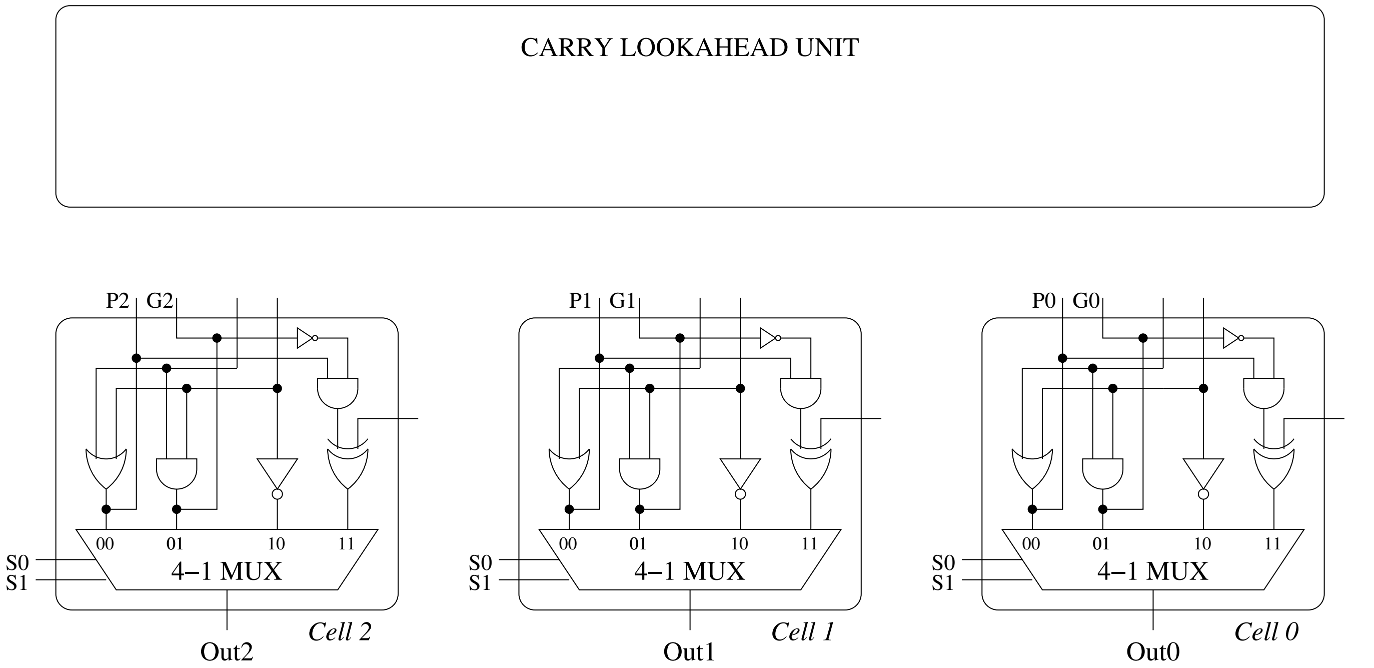


[8] The diagram above shows 3 ALU cells and a Carry Lookahead Unit. In the above diagram, draw lines to show how you would connect this up to provide a 3-bit ALU which uses Carry Lookahead. Label all the (unlabled) inputs and outputs accordingly. (You do not have to draw what is inside the Carry Lookahead Unit, and you do not need to worry about subtraction for this problem.)

[3] Write the equation for the carry into Cell 2 in this ALU, in terms of P's and G's.

Assuming all input signals arrive at time zero, and

NOT gate takes 2 time units	XOR gate takes 5 time units
OR gate takes 3 time units	CL unit takes 7 time units
AND gate takes 4 time units	MUX takes 10 time units

[4] What is the worst case time to create Out0?

[2] How long to create Cin1?

[3] How long to create Out2?

[8] A coin-operated machine takes Ones (X1) and Twos (X2), and dispenses merchandise (Z1=1) when the sum of the inputs is greater than or equal to Three (One + Two = Three, astonishingly). Only 1 coin can be input at a time. The machine should give exact change.

Using a Mealy model, draw the State Transistion Diagram (the circles and the arcs) for this finite state machine. Label the transitions on the diagram using the format we used in class (inputs over outputs). Let state S0=no money input (the Start state). Input 00 sends you back to the same state you are in.

[8] Draw the state transition diagram for a 2-bit saturating up/down counter that counts in gray code fashion (00, 01, 11, 10). 00 is the lowest possible, 10 is the highest. Assume input X=1 indicates count up, and X=0 indicates count down. Use a Moore model.

13] (12) On a planet far, far away you have been hired to create a vending machine which accepts two coins, the 1 Quatloo piece and the 3 Quatloo piece. Merchandise costs 4 Quatloos, and the machine must give exact change. Let X1=3 Quatloo coin and X2=1 Quatloo coin, and assume both coins cannot be inserted simultaneously.

Using a Mealy model, draw the State Transistion Diagram (the circles and the arcs) for this finite state machine. Label the transitions on the diagram using the format we used in class (inputs over outputs). You must clearly show what each output bit represents. Let state S0=no money input (the Start state). Input 00 sends you to the same state you are currently in.

In this question, we are going to wire up an 8-bit machine. This machine will use a 1-operand format, meaning that instructions are of the type ACC=ACC+R. So, for example, "Add R" is ACC=ACC+R.

The machine is byte-addressable. Immediates are sign-extended, and jumps are done by adding the sign-extended Immediate field to the PC.

The machine has 2 different instruction formats: A and B.

A-type:	Opcode	extra
	7-4	3-0

B-type:	Opcode	Immediate
	7-4	3-0

The ALU can perform 7 functions, written this way: OP [ALU2 ALU1 ALU0]

Operation	**ALU2**	**ALU1**	**ALU0**
Add	0	0	1
Sub	0	1	0
Increment B	0	1	1
AND	1	0	0
OR	1	0	1
NOT A	1	1	0
XOR	1	1	1

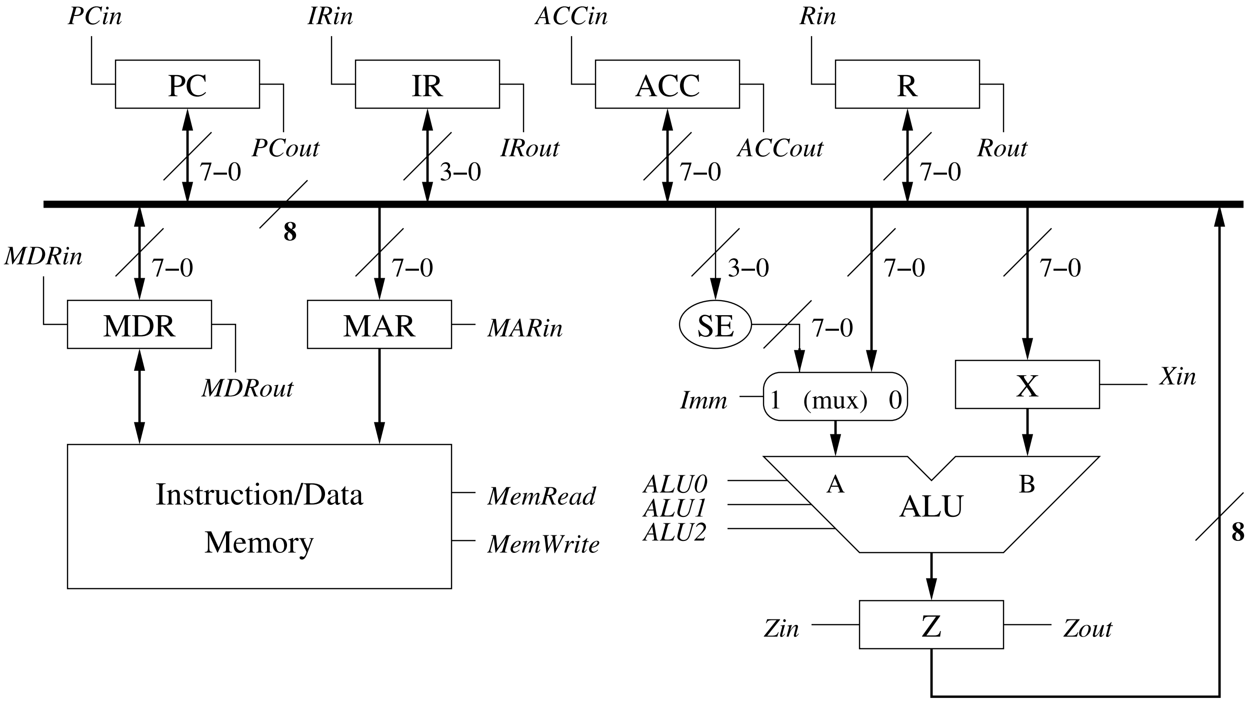
Here are a few of the instructions that have been defined:

Name	**Opcode**	**Operation**
ADD	0001	ACC=ACC+R
ADDM	0010	ACC=ACC+MEM[R]
DDI	0011	ACC=ACC+Imm
COPYA2R	0100	R=ACC
COPYR2A	0101	ACC=R
LOADACC	0110	ACC=MEM[R]
JMP	1111	PC=PC+Imm

On the following page is a diagram of the machine. The control signals are in italics. The sign extend logic creates a 5-bit value which matches the contents of bit 3, so that the 8-bit value generated looks like 33333210 (instead of 76543210).

Here are the 20 control signals.

PCin	PCout	IRin	IRout	Rin	Rout	MARin
ACCin	ACCout	MDRin	MDRout	Zin	Zout	Xin
ALU0	ALU1	ALU2	MemRead	MemWrite	Imm	



20] [10] Fill in the steps necessary to perform an instruction fetch (incrementing the PC is considered part of the instruction fetch process). Assume memory can respond in a single cycle.

Step	PCin	IRin	Rin	ACCin	MARin	MDRin	Zin	Xin	PCout	IRout	Rout	ACCout	MDRout	Zout	ALU2	ALU1	ALU0	Mread	Mwrite	Imm
0																				
1																				
2																				
3																				
4																				
5																				

21] [8] Now that you have done the instruction fetch, fill in the microcode steps necessary to perform the following instruction: **JMP -3**

Step	PCin	IRin	Rin	ACCin	MARin	MDRin	Zin	Xin	PCout	IRout	Rout	ACCout	MDRout	Zout	ALU2	ALU1	ALU0	Mread	Mwrite	Imm
0																				
1																				
2																				
3																				
4																				

In this question, we are going to wire up an 8-bit machine. This machine will use a 1-operand format, meaning that instructions are of the type ACC=ACC+R. So, for example, "Add R" is ACC=ACC+R.

The machine is byte-addressable. Immediates are sign-extended, and jumps are done by adding the sign-extended Immediate field to the PC.

The machine has 2 different instruction formats: A and B.

A-type:	Opcode	extra
	7-4	3-0
B-type:	Opcode	Immediate
	7-4	3-0

The ALU can perform 4 functions, written this way: OP [ALU1 ALU0]

Operation	**ALU1**	**ALU0**
And	0	0
Or	0	1
Not A	1	0
Add	1	1

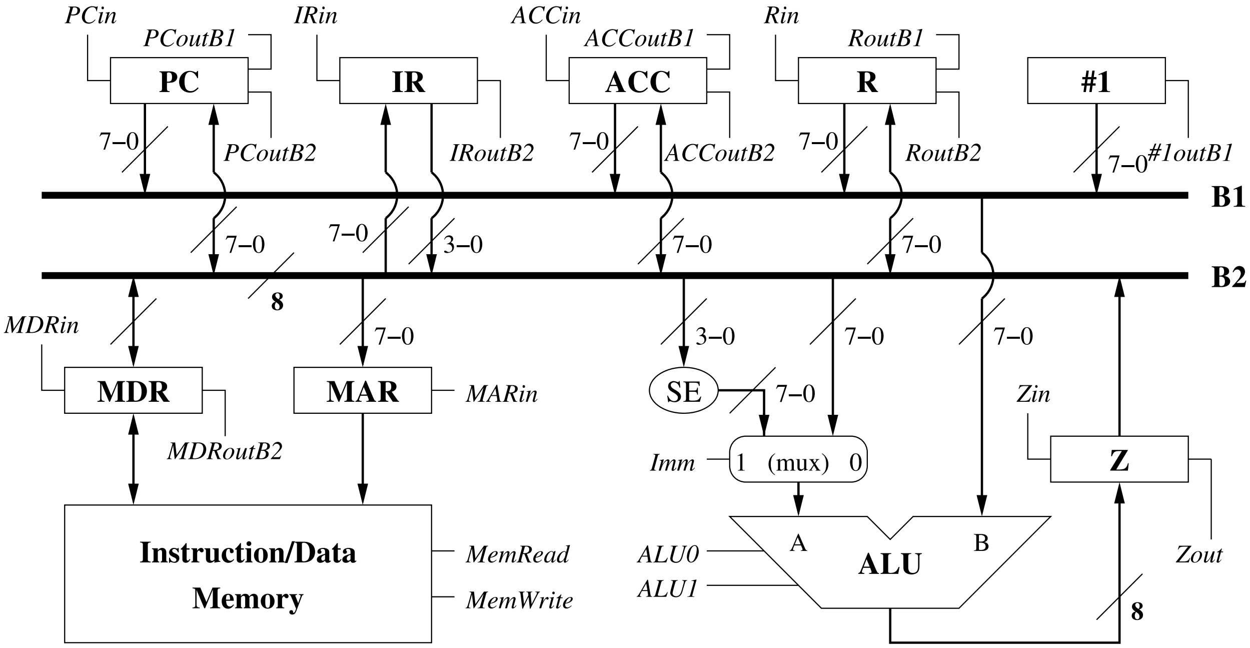
Here are a few of the instructions that have been defined:

Name	**Opcode**	**Operation**
ADD	0001	ACC=ACC+R
ADDM	0010	ACC=ACC+MEM[R]
ADDI	0011	ACC=ACC+Imm
COPYA2R	0100	R=ACC
COPYR2A	0101	ACC=R
LOADACC	0110	ACC=MEM[R]
JMP	1111	PC=PC+Imm

On the following page is a diagram of the machine. The control signals are in italics. The sign extend logic creates a 5-bit value which matches the contents of bit 3, so that the 8-bit value generated looks like 33333210 (instead of 76543210).

Here are the 22 control signals.

PCin	PCoutB1	PCoutB2	IRin	IRoutB2	#1outB1
Rin	RoutB1	RoutB2	ACCin	ACCoutB1	ACCoutB2
MDRin	MDRoutB2	Zin	ZoutB2	MARin	
ALU0	ALU1	MemRead	MemWrite	Imm	



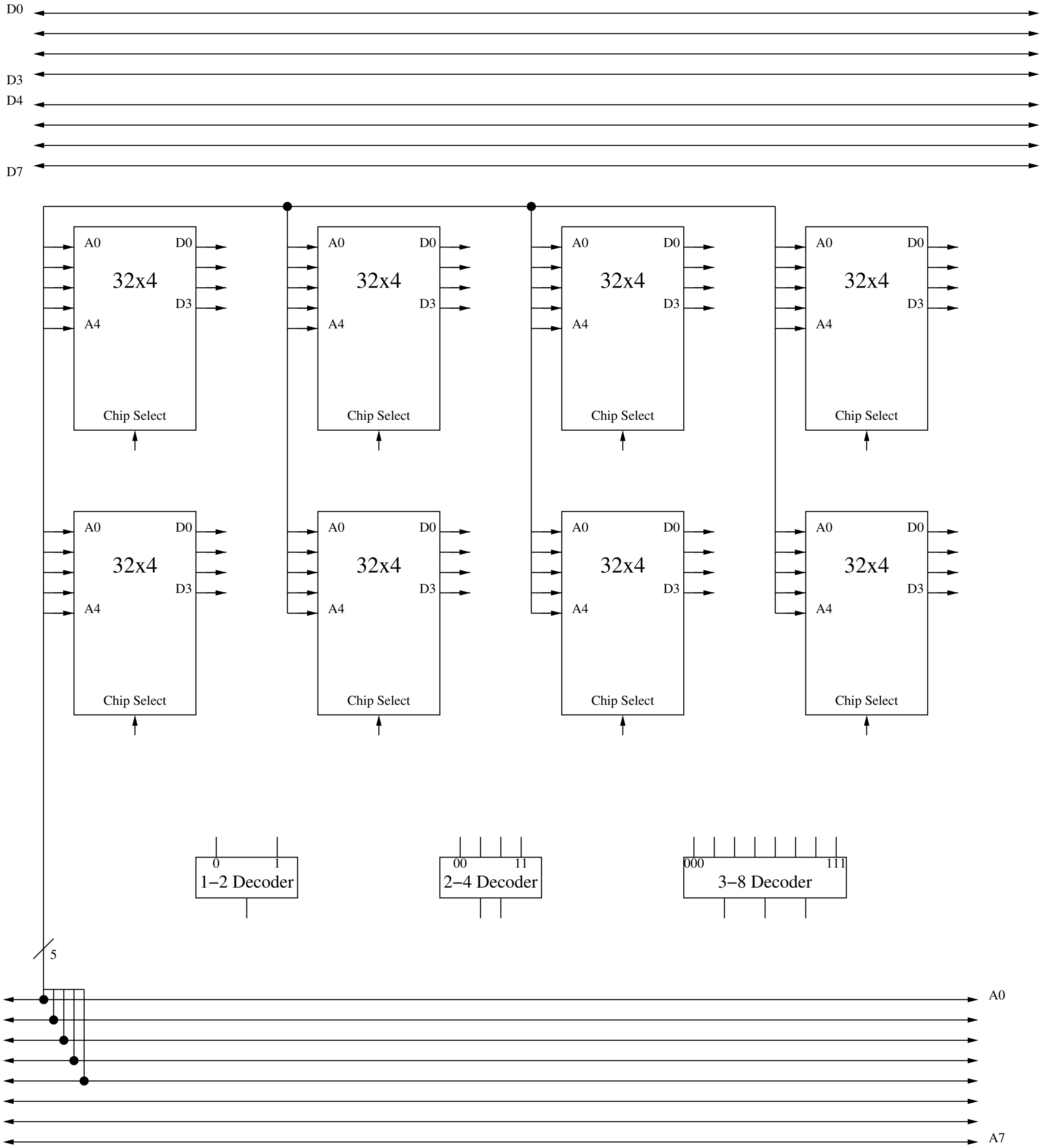
[10] Fill in the steps necessary to perform an instruction fetch (incrementing the PC is considered part of the instruction fetch process). Assume memory can respond in a single cycle.

Step	PCin	IRin	ACCin	Rin	MDRin	MARin	Zin	PCoutB1	PCoutB2	IRoutB2	ACCoutB1	ACCoutB2	RoutB1	RoutB2	#1outB1	MDRoutB2	ZoutB2	ALU1	ALU0	Mread	Mwrite	Imm
0																						
1																						
2																						
3																						
4																						

[7] Now that you have done the instruction fetch, fill in the microcode steps necessary to perform the following instruction: **ADDI 5**

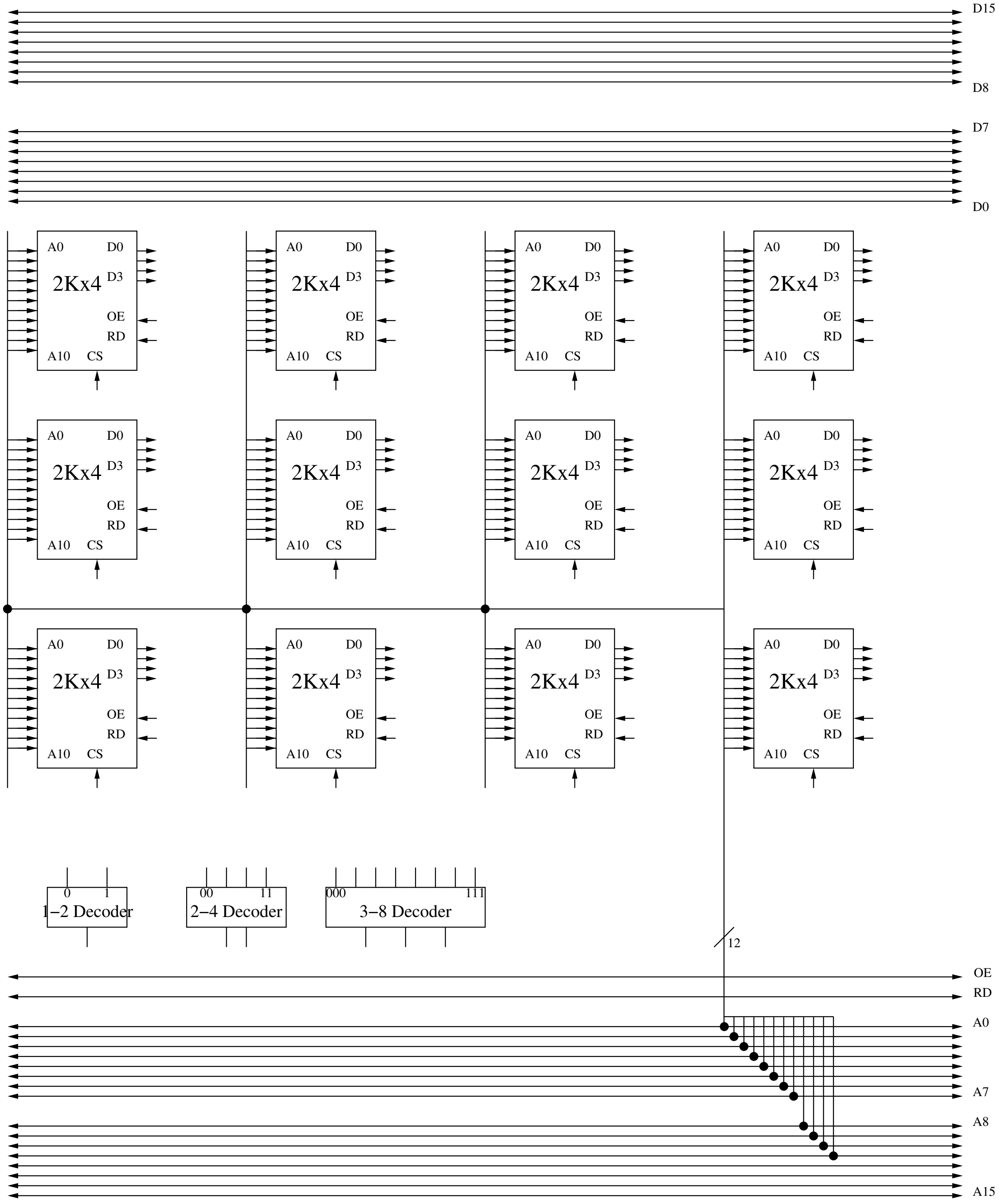
Step	PCin	IRin	ACCin	Rin	MDRin	MARin	Zin	PCoutB1	PCoutB2	IRoutB2	ACCoutB1	ACCoutB2	RoutB1	RoutB2	#1outB1	MDRoutB2	ZoutB2	ALU1	ALU0	Mread	Mwrite	Imm
0																						
1																						
2																						
3																						

[8] Add the connections to the following diagram necessary to create a 128 byte memory (reminder - a "32x4" chip has 32 entries, each 4 bits wide). Not all of the hardware shown is required to perform this task.



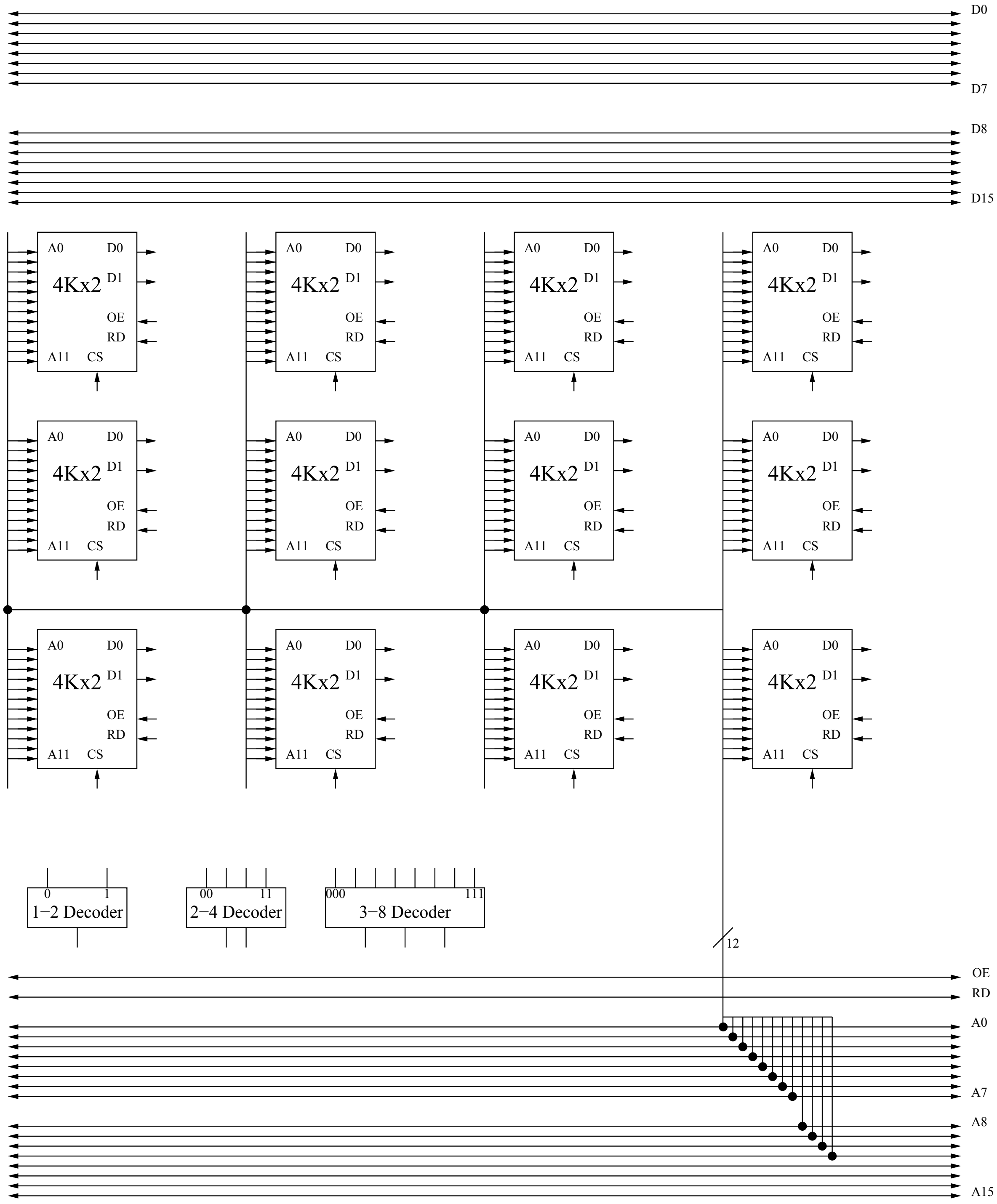
13. (8) Add the connections to the following diagram necessary to create a 8Kx8 memory. Not all of the hardware shown is required to perform this task.

CS - Chip Select
OE - Output Enable
RD - Read (Read/Write, technically)



13. (8 pts) Add the connections to the following diagram necessary to create a 8Kx8 memory. Not all of the hardware shown is required to perform this task.

CS - Chip Select
OE - Output Enable
RD - Read (Read/Write, technically)



In this question we are dealing with a Direct Mapped cache. Assume 8-bit addresses are partitioned in the following manner:

Tag	Entry	Offset
TTTT	EEE	L

(Tag is left most 4 bits, entry is middle 3, offset into line is right most bit)

You are given the following address reference sequence (in Hex):

0x43,0x42,0x44,0x93,0x45

[10] In the table below, fill in the Cache's Tag values after each memory reference has been processed. If it is a miss, you should enter what the new tag should be in the appropriate slot. (X indicates the entry is invalid). If it is a hit, you should place an H in the correct location. There may be more Tag Array entries than you need. *Only write down things that change - do not fill in all the entries that remain the same.*

Tag Array	Contents of Tag Array after processing address (Time ->)					
Entry Number	Initial Contents	0x43 (0 1 0 0 0 0 1 1)	0x42 (0 1 0 0 0 0 1 0)	0x44 (0 1 0 0 0 1 0 0)	0x93 (1 0 0 1 0 0 1 1)	0x45 (0 1 0 0 0 1 0 1)
0	X					
1	X					
2	X					
3	X					
4	X					
5	X					
6	X					
7	X					
8	X					
9	X					
10	X					
11	X					
12	X					
13	X					
14	X					
15	X					

[1] What set of memory addresses are sent to memory on the first miss?

14. (13) Assume a byte-addressable computer with a 32-bit word size and 256 bytes of memory. This machine has a 64-byte physically addressed Direct-Mapped cache with a line size of 1 word and an access time of 1 cycle. Given the following address reference sequence (in Hex):

0xB5,0xB7,0x37,0x38,0xB6

a) Write down how you are partitioning each address (which bits are the Tag, offset, etc.)

b) In the table below, fill in the proposed Cache's Tag values after each memory reference has been processed, and circle H if it is a hit or M if it is a miss. If it is a miss, you should also enter what the new tag should be. (X indicates the entry is invalid). There may be more Tag Array entries than you need. *Only write down things that change - do not fill in all the entries that remain the same.*

Tag Array	Contents of Tag Array after processing address (Time ->)					
Entry Number	Initial Contents	0xB5 (10110101)	0xB7 (10110111)	0x37 (00110111)	0x38 (00111000)	0xB6 (10110110)
0	X					
1	X					
2	X					
3	X					
4	X					
5	X					
6	X					
7	X					
8	X					
9	X					
10	X					
11	X					
12	X					
13	X					
14	X					
15	X					
		H / M	H / M	H / M	H / M	H / M

What set of memory addresses are sent to memory on the first miss?

In this question we are dealing with a Direct Mapped cache. Assume 8-bit addresses are partitioned in the following manner:

Tag	Entry	Offset
TTTT	EEE	L

(Tag is left most 4 bits, entry is middle 3, offset into line is right most bit)

You are given the following address reference sequence (in Hex):

0x47,0x46,0x44,0x93,0x45

23] [10] In the table below, fill in the Cache's Tag values after each memory reference has been processed. If it is a miss, you should enter what the new tag should be in the appropriate slot. (X indicates the entry is invalid). If it is a hit, you should place an H in the correct location. There may be more Tag Array entries than you need. *Only write down things that change - do not fill in all the entries that remain the same.*

Tag Array	Contents of Tag Array after processing address (Time ->)					
Entry Number	Initial Contents	0x47 (0 1 0 0 0 1 1 1)	0x46 (0 1 0 0 0 1 1 0)	0x44 (0 1 0 0 0 1 0 0)	0x93 (1 0 0 1 0 0 1 1)	0x45 (0 1 0 0 0 1 0 1)
0	X					
1	X					
2	X					
3	X					
4	X					
5	X					
6	X					
7	X					
8	X					
9	X					
10	X					
11	X					
12	X					
13	X					
14	X					
15	X					

24] [2] What set of memory addresses are sent to memory on the first miss?

14. (13) Assume a byte-addressable computer with a 32-bit word size and 256 bytes of memory. In this machine accessing main memory takes 10 clock cycles (in addition to the time necessary to do a cache lookup), and the bus between main memory and the processor is 16-bits wide. This machine also has a 64-byte physically addressed Direct-Mapped cache with a line size of 2 words and an access time of 1 cycle. Given the following address reference sequence (in Hex):

0xB5,0xB7,0x37,0x38,0x39

a) Write down how you are partitioning each address (which bits are the Tag, offset, etc.)

b) In the table below, fill in the proposed Cache's Tag values after each memory reference has been processed. If it is a hit, mark the entry number to indicate this, and if it is a miss enter what the new tag should be. (X indicates the entry is invalid). There may be more Tag Array entries than you need.

Tag Array	Contents of Tag Array after processing address (Time ->)					
Entry Number	Initial Contents	0xB5 (10110101)	0xB7 (10110111)	0x37 (00110111)	0x38 (00111000)	0x39 (00111001)
0	X					
1	X					
2	X					
3	X					
4	X					
5	X					
6	X					
7	X					
8	X					
9	X					
10	X					
11	X					
12	X					
13	X					
14	X					
15	X					

What set of memory addresses are sent to memory on the first miss?

[6] Assume an 8-bit processor, with a 24-byte 3-way set associative cache and a linesize of 2. This is the contents of the cache (as always, there may be more information than you need):

Set 0				
Tag	Data (0)	Data (1)	Data (2)	Data (3)
10110	0x3b	0xC1	0x43	0x86
11001	0x9D	0x90	0xB3	0x65
00110	0x9F	0xA8	0x28	0x54

Set 1				
Tag	Data (0)	Data (1)	Data (2)	Data (3)
00010	0xd8	0xA9	0x85	0x94
10110	0xD4	0x9B	0xEA	0xD1
11011	0x2A	0xF6	0x80	0xE0

Set 2				
Tag	Data (0)	Data (1)	Data (2)	Data (3)
11110	0x72	0x9A	0x49	0x6f
11011	0xC4	0x25	0xC8	0x2E
00111	0x69	0x83	0x1A	0x94

Set 3				
Tag	Data (0)	Data (1)	Data (2)	Data (3)
11111	0x4A	0xF7	0x24	0xB4
01011	0x94	0xFA	0x92	0x48
00100	0x3C	0xD8	0x6F	0xCD

a) If the processor issues the address

1 0 1 1 0 1 1 0

Is this a hit in the cache? (YES NO) If YES, circle the entry and the data value that is returned.

b) If the processor issues the address

1 1 0 1 1 0 1 1

Is this a hit in the cache? (YES NO) If YES, circle the entry and the data value that is returned.

15. (13) What if a 32-byte 2-way Set Associative Cache with a line size of 2 words is used instead of the Direct-mapped Cache? Remember, this is a byte-addressable machine with a 32-bit word size and 256 bytes of memory. Given the same address reference sequence (in Hex) as before:

0xB5,0xB7,0x37,0x38,0xB6

a) Write down how you are partitioning each address (which bits are the Tag, offset, etc.)

b) In the table below, fill in the proposed Cache's Tag values after each memory reference has been processed. If the reference is a hit, put an "H" in the tag field, and if it is a miss write down what the new tag should be. Use an LRU replacement scheme, and after each address is processed be sure to indicate the age of the references. There may be more entries than you need. MRU = Most Recently Used, LRU = Least Recently Used. *Only write down things that change - do not fill in all the entries that remain the same.*

Tag Array				Contents of Tag Array after processing address (Time ->)									
Set #	Entry #	Initial contents		0xB5 (10110101)		0xB7 (10110111)		0x37 (00110111)		0x38 (00111000)		0xB6 (10110110)	
		Age	Tag	Age	Tag	Age	Tag	Age	Tag	Age	Tag	Age	Tag
0	0	MRU	0 0 1 1										
	1	LRU	1 1 1 0										
	2		1 0 0 0										
1	0		0 1 0 0										
	1	MRU	0 0 1 1										
	2	LRU	1 1 0 0										
2	0	LRU	1 1 0 0										
	1		0 1 0 1										
	2	MRU	0 1 1 0										
3	0	LRU	0 0 1 0										
	1		0 1 1 1										
	2	MRU	0 1 1 0										

15. (13) What if a 48-byte 3-way Set Associative Cache (instead of the Direct-mapped Cache) with a line size of 1 word is used instead? Remember, this is a byte-addressable machine with a 32-bit word size, a 16-bit bus between processor and memory, and a Main Memory access time of 10 cycles (in addition to the time necessary to to a cache lookup). The Cache access time is still 1 cycle. Given the same address reference sequence (in Hex) as before:

0xB5,0xB7,0x37,0x38,0x39

a) Write down how you are partitioning each address (which bits are the Tag, offset, etc.)

b) In the table below, fill in the proposed Cache's Tag values after each memory reference has been processed. If it is a hit, put an "H" in the tag field, and if it is a miss write down what the new tag should be. Use an LRU replacement scheme, and after each address is processed be sure to indicate the age of the references. There may be more entries than you need. MRU = Most Recently Used, LRU = Least Recently Used.

Tag Array				Contents of Tag Array after processing address (Time ->)									
Set #	Entry #	Initial contents		0xB5 (10110101)		0xB7 (10110111)		0x37 (00110111)		0x38 (00111000)		0x39 (00111001)	
		Age	Tag	Age	Tag	Age	Tag	Age	Tag	Age	Tag	Age	Tag
0	0	MRU	0 0 1 1										
	1	LRU	1 1 1 0										
	2		1 0 0 0										
1	0		0 1 0 0										
	1	MRU	0 0 1 1										
	2	LRU	1 1 0 0										
2	0	LRU	1 1 0 0										
	1		0 1 0 1										
	2	MRU	0 1 1 0										
3	0	LRU	0 0 1 0										
	1		0 1 1 1										
	2	MRU	0 1 1 0										

22] [6] Assume a 10-bit processor, with a 64-byte 2-way set associative cache and a linesize of 8. This is the contents of the cache (as always, there may be more information than you need):

Set 0								
Tag	D(0)	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)
10101	3B	C1	43	86	3B	C1	43	86
11001	9D	90	B3	65	F4	EB	7A	BC

Set 1								
Tag	D(0)	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)
11110	72	9A	49	6F	84	CC	62	8D
11011	C4	25	C8	2E	75	E0	5A	F5

Set 2								
Tag	D(0)	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)
11110	E9	BA	C6	03	B8	AB	14	2B
11011	17	09	5A	8B	8F	0D	25	CD

Set 3								
Tag	D(0)	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)
11110	F8	E5	B4	AB	F4	50	6B	07
11011	EF	8C	99	9E	71	46	BF	0F

Set 7								
Tag	D(0)	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)
00010	03	47	05	45	E8	39	39	9D
10101	51	FE	CF	B5	5D	2A	DE	D8

Set 6								
Tag	D(0)	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)
11111	5A	70	5C	18	15	52	ED	1C
01011	CF	7E	2E	F9	25	8C	9C	38

Set 5								
Tag	D(0)	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)
11110	E0	C4	2C	B4	5D	AE	66	6E
11011	FF	D0	81	2E	4E	94	E5	20

Set 4								
Tag	D(0)	D(1)	D(2)	D(3)	D(4)	D(5)	D(6)	D(7)
11110	F7	8B	4E	E6	E7	94	B9	2D
11011	F9	09	23	A1	7A	C5	65	35

a) If the processor issues the address

1 0 1 0 1 1 0 1 1 0

Is this a hit in the cache? (YES NO) If YES, circle the entry and the data value that is returned.

b) If the processor issues the address

1 1 0 1 1 1 1 1 1 0

Is this a hit in the cache? (YES NO) If YES, circle the entry and the data value that is returned.

In this problem we have a machine that generates 12 bit Virtual Addresses, uses 256 byte pages, and has 2K bytes of memory. The TLB has 3 entries, and is fully associative.

The first address the CPU issues is

1 0 1 0 0 0 0 1 1 0 0 1 (0xA19)

The requested page is not currently resident in the memory, so a page fault is generated and the Operating System is called in. After consulting its internal data structures, it decides to put the requested page in frame number 3.

Here are the contents of the page table and the TLB before the address is sent to memory:

Page Table

Entry	Contents	Valid
0000		N
0001		N
0010		N
0011		N
0100		N
0101		N
0110		N
0111		N
1000		N
1001		N
1010		N
1011		N
1100		N
1101		N
1110		N
1111		N

TLB

Tag	Contents	Valid
		N
		N
		N

[2] What is the physical address (in binary) that gets sent to memory?

[2] Update the page table to show what it contains after the physical address is generated.

[3] Update the TLB to show what it contains after the physical address is generated.

Now, assume the following address is generated next:

1 0 1 0 0 1 1 1 1 0 1 0

[1] What physical address gets sent to memory?

The following tables contain **some** of the information about a segmented, paged virtual memory system and certain select memory locations. Total physical memory size is 8K bytes, and the page size is 256 bytes. All numbers in this table are in decimal unless otherwise noted.

Segment Table		
Entry Number	Presence Bit	Page Table
0	1	5
1	1	2
2	1	0
3	0	7
4	1	5
5	1	3
6	1	1
7	0	4

Page Table 0			
Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
0	1	1234123	0x4
2	0	0893748	0x7
4	1	2489567	0x1
8	1	9623873	0x17
16	1	B0F6BD3	0x23
25	0	32829AA	0xA
29	1	56D87AC	0xC
31	1	10A876D	0x6

Page Table 2			
Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
0	1	1234123	0xF
1	0	0893748	0x11
2	1	2489567	0x14
3	1	9623873	0x27
4	1	BC56BD3	0x29
6	0	832759E	0x15
10	1	46B37AC	0x24
31	1	810476D	0x16

Memory	
Address	Contents
0x01A4	0x76
0x04A4	0x73
0x06A4	0x32
0x10A4	0x46
0x17A4	0x30
0x20A4	0xa9
0x21A4	0x05
0x24A4	0x74
0x26A4	0x29

Page Table 5			
Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
1	1	1234123	0xD
3	0	0893748	0x13
9	0	2489567	0x19
15	1	9623873	0x20
18	1	AE76BD3	0x18
22	0	328759A	0xE
25	1	11D87BE	0x12
31	1	91C875D	0x0

Page Table 7			
Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
0	1	1234123	0x5
1	0	0893748	0x26
2	1	2489567	0x21
3	1	9623873	0x2
4	1	AE76BD3	0x1A
5	1	328759A	0x10
6	1	56D87AC	0x3
7	1	10A876D	0x8

[9] For each of the following convert the virtual address into a physical address (if possible) and write down the value of the memory location corresponding to the address. If it is not possible to do so, explain why.

0x8FA4 (**1 0 0 0 1 1 1 1 1 0 1 0 0 1 0 0** in binary).

0x03A4 (**0 0 0 0 0 0 1 1 1 0 1 0 0 1 0 0** in binary).

0x6FA4 (**0 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0** in binary).

16. (10) The following tables contain **some** of the information about a segmented, paged virtual memory system and certain select memory locations. Total physical memory size is 16K bytes, and the page size is 256 bytes. All numbers in this table are in decimal unless otherwise noted.

Segment Table		
Entry Number	Presence Bit	Page Table
0	1	5
1	1	2
2	1	0
3	0	7
4	1	5
5	1	3
6	1	1
7	0	4

Page Table 0			
Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
0	1	1234123	0x4
2	0	0893748	0x7
4	1	2489567	0x1
8	1	9623873	0x17
16	1	B0F6BD3	0x23
25	0	32829AA	0xA
29	1	56D87AC	0xC
31	1	10A876D	0x6

Page Table 2			
Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
0	1	1234123	0xF
1	0	0893748	0x11
2	1	2489567	0x14
3	1	9623873	0x27
4	1	BC56BD3	0x29
6	0	832759E	0x15
10	1	46B37AC	0x24
31	1	810476D	0x16

Memory	
Address	Contents
0x01A4	0x76
0x04A4	0x73
0x06A4	0x32
0x10A4	0x46
0x17A4	0x30
0x20A4	0xa9
0x21A4	0x05
0x24A4	0x74
0x26A4	0x29

Page Table 5			
Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
1	1	1234123	0xD
3	0	0893748	0x13
9	0	2489567	0x19
15	1	9623873	0x20
18	1	AE76BD3	0x18
22	0	328759A	0xE
25	1	11D87BE	0x12
31	1	91C875D	0x0

Page Table 7			
Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
0	1	1234123	0x5
1	0	0893748	0x26
2	1	2489567	0x21
3	1	9623873	0x2
4	1	AE76BD3	0x1A
5	1	328759A	0x10
6	1	56D87AC	0x3
7	1	10A876D	0x8

For each of the following convert the virtual address into a physical address (if possible) and write down the value of the memory location corresponding to the address. If it is not possible to do so, explain why.

0x2AA4 (**0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 0** in binary).

0x48A4 (**0 1 0 0 1 0 0 0 1 0 1 0 0 1 0 0** in binary).

0x89A4 (**1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0** in binary).

0xEFA4 (**1 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0** in binary).

16. (8 pts) The following tables contain some of the information about a segmented, paged virtual memory system and certain select memory locations. Total physical memory size is 32K bytes, and the page size is 512 bytes. All numbers in this table are in decimal unless otherwise noted. (**Note:** The maximum number of entries in the page tables is significant, but the number of entries in the Segment table is not.)

Segment Table

Entry Number	Presence Bit	Page Table
0	1	5
1	1	2
2	1	0
3	0	7
7	1	5
12	1	3
13	1	1
15	1	4

Page Table 0

Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
0	1	1234123	0x4
2	0	0893748	0x7
4	1	2489567	0x1
8	1	9623873	0x17
16	1	B0F6BD3	0x23
25	0	32829AA	0xA
29	1	56D87AC	0xC
31	1	10A876D	0x6

Page Table 2

Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
0	1	1234123	0xF
1	0	0893748	0x11
2	1	2489567	0x14
3	1	9623873	0x27
4	1	BC56BD3	0x29
6	0	832759E	0x15
10	1	46B37AC	0x24
31	1	810476D	0x16

Memory

Address	Contents
0x00A4	0x76
0x01A4	0x73
0x02A4	0x32
0x03A4	0x46
0x04A4	0x30
0x06A4	0xa9
0x0AA4	0x05
0x31A4	0x74
0x62A4	0x29

Page Table 5

Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
1	1	1234123	0xD
3	0	0893748	0x13
9	0	2489567	0x19
15	1	9623873	0x20
18	1	AE76BD3	0x18
22	0	328759A	0xE
25	1	11D87BE	0x12
31	1	91C875D	0x0

Page Table 7

Entry Number	Present? (1=Yes)	Disk Addr	Frame Number
0	1	1234123	0x5
1	0	0893748	0x26
2	1	2489567	0x21
3	1	9623873	0x2
4	1	AE76BD3	0x1A
5	1	328759A	0x10
6	1	56D87AC	0x3
7	1	10A876D	0x8

For each of the following convert the virtual address into a physical address (if possible) and write down the value of the memory location corresponding to the address. If it is not possible to do so, explain why.

0x3EA4 (**0 0 1 1 1 1 1 0 1 0 1 0 0 1 0 0** in binary).

0x4CA4 (**0 1 0 0 1 1 0 0 1 0 1 0 0 1 0 0** in binary).

0x89A4 (**1 0 0 0 1 0 0 1 1 0 1 0 0 1 0 0** in binary).

0xEFA4 (**1 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0** in binary).

Here is a code sequence. (Remember, operands are written Rdst, Rsrc1, Rsrc2)

```
add     R2, R4, R4


and     R3, R1, R2


lw      R1, 8(R10)


sub     R2, R4, R1
```

Assuming a standard 5-stage pipeline that does not support hazard detection and does no forwarding,

[2] Indicate all read after write dependencies (draw lines/arrows between them).

[4] Insert as many No Operations (NOPS) as required in order to ensure this code runs correctly. (Remember, writes to the register file occur on the first half of the cycle, and reads occur during the second half).

[2] Circle the NOPs that can be removed if forwarding and hazard detection logic is implemented.

Potentially useful information

F	D	E	M	W						
	F	D	E	M	W					
		F	D	E	M	W				
			F	D	E	M	W			
				F	D	E	M	W		
					F	D	E	M	W	
						F	D	E	M	W

2^4 = 16	0x1 = 0001
2^8 = 256	0x2 = 0010
2^10 = 1K	0x4 = 0100
2^20 = 1M	0x8 = 1000
2^30 = 1G	