University of California, Davis
Department of Computer Science

ECS-154A Computer Architecture

Professor Farrens

Winter 2024

Final

Cheating on exams is prohibited by the Academic Code of Conduct. I understand that if I am caught cheating on this exam my scores will be negated and I will receive no credit on the exam. Persons caught cheating will also face University disciplinary actions.

Name: _____

Signature: _____

Student ID number: _____

---

[9] **Circle the right answer:**

The contents of Volatile memory (remain / disappear) after the power is removed.

Static RAM is (hotter / cooler) than Dynamic RAM.

Write-through caches tend to use (write-allocate / no-write-allocate) policies.

Dynamic RAM is built out of (flip flops / capacitors).

A flip-flop (is / is not) the same as a gated latch.

Cache coherence (is / is not) only a problem for parallel processors.

The goal of the memory hierarchy is to (increase / decrease) the amount of fast memory.

Electrons flowing through wires and transistors (generates / does not generate) heat.

Virtual Memory (is / is not) a mapping from one address space to another.

Very short answer questions (1-5 words):

[2] Caches, Virtual Memory, and in fact the entire memory hierarchy is possible because programs exhibit what principle?

[2] What structure uses a dirty bit?

[2] What is one of the simplest (and oldest) techniques for exploiting parallelism among instructions?

[2] Given the bit pattern 01001, if you want to add a parity bit and use odd parity, what should the bit be set to?

[3] What is the 3-term CPU time equation?

[2] When dealing with a write to a cache that hits, what are the two possible approaches?

[2] What are the two main ways to define performance?

Short answer questions (5-15 words):

[2] Explain what the hold time is for a flip-flop.

[2] Explain what the setup time is for a flip-flop.

[3] What is the difference between a latch and a flip-flop?

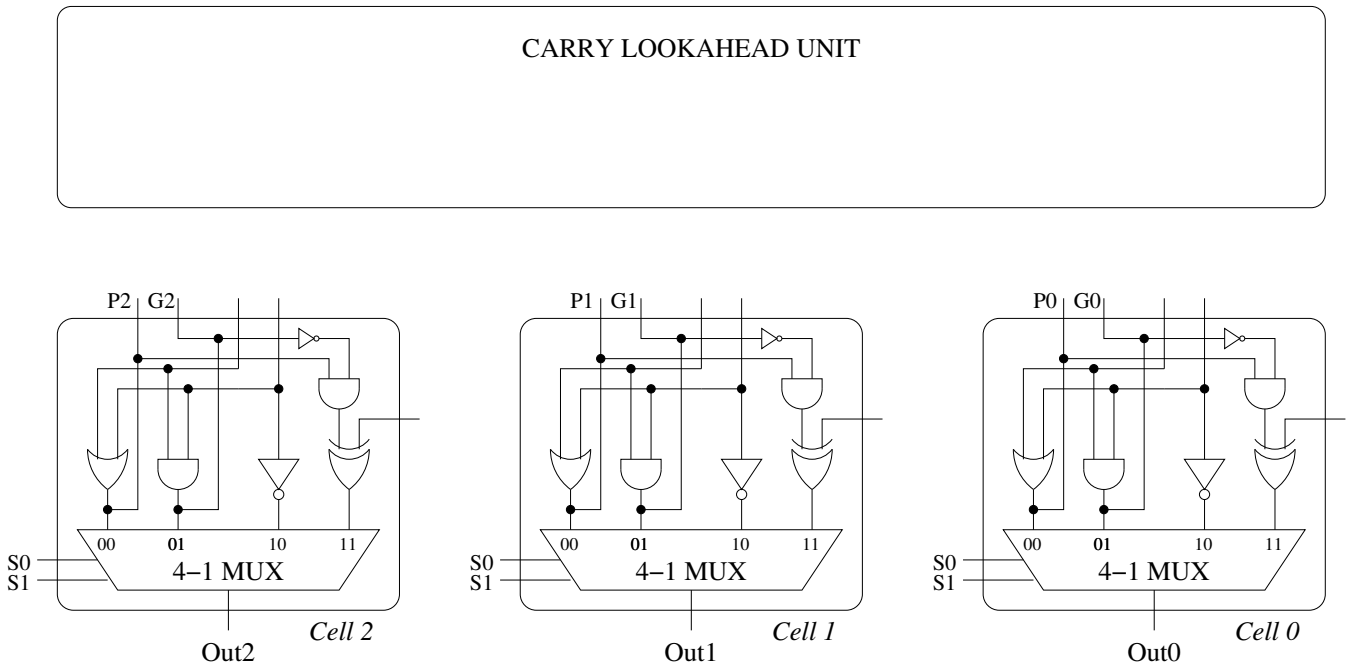
[3] The multicycle design uses a single memory, but the single cycle design has two. Why is that?

[3] What is leakage current?

[6] In your first design of a 5-stage pipeline (F,D,E,M,W), F takes 16 time units, D takes 17, E takes 30, M takes 31, and W takes 29.

a) What will the clock cycle time be for this pipeline?

b) Is it a balanced pipeline? If not, explain what you could do to fix it. What would the cycle time be now?



[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
OR gate takes 3 time units
AND gate takes 4 time units
XOR gate takes 5 time units
CL unit takes 7 time units
MUX takes 10 time units

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

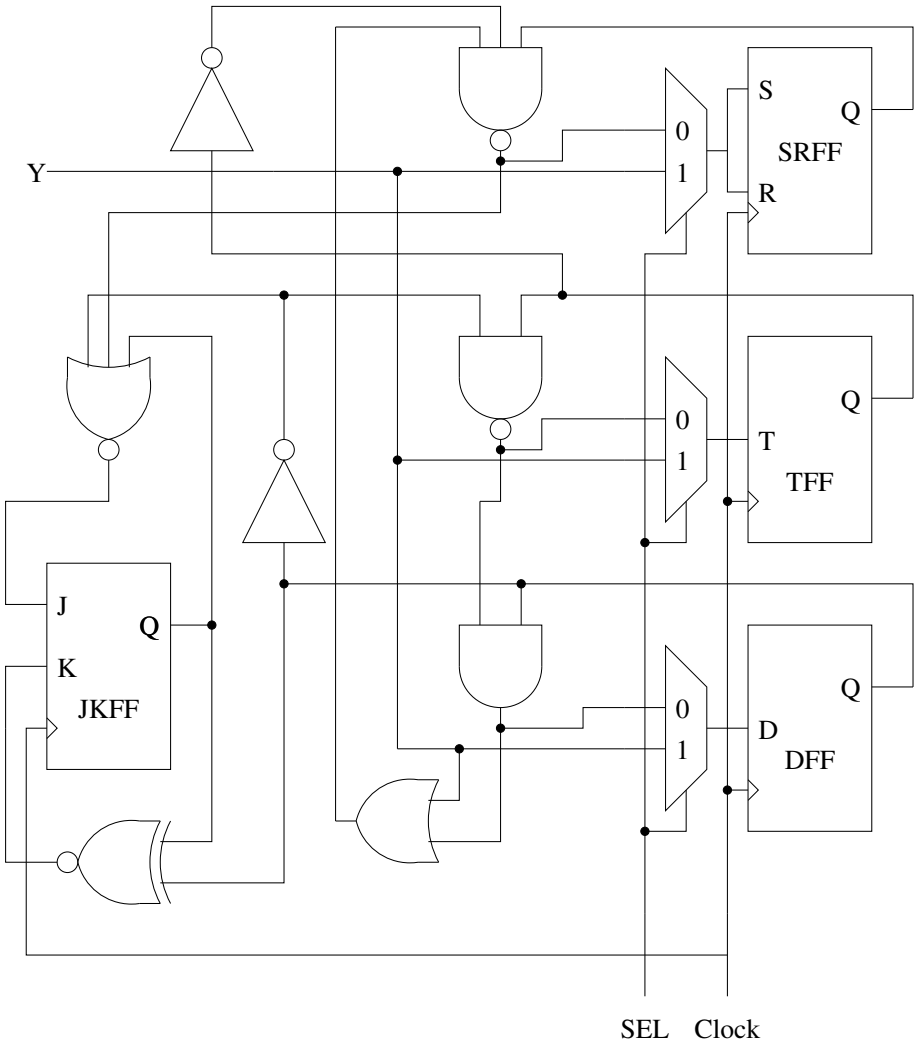
[2] How long to create Cin1?

[3] How long to create Out2?

[10] Assuming rising edge-triggered flipflops, what is the minimum cycle time (the minimum time between rising clock edges) that will still guarantee correct behavior for the following circuit? Use the following delay values, and assume all input signals become valid at time 0. (Tprop is the propagation time for the flipflop, the time it takes from the rising edge of the clock until the output of the FF is valid.)

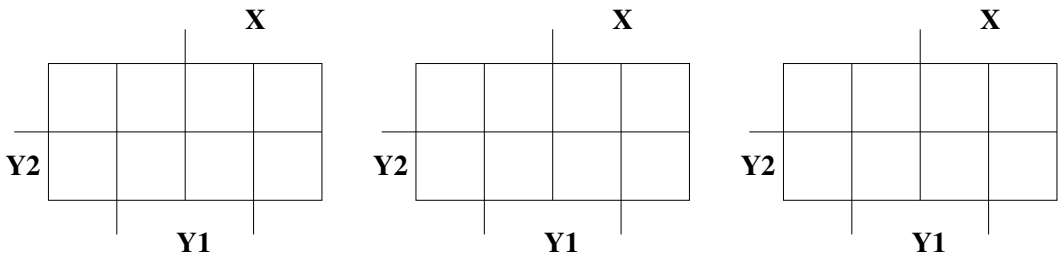
AND: 4ns OR: 3ns NAND: 6ns NOR: 5ns NOT: 1ns MUX: 5ns XNOR: 22ns
Tprop (TFF): 11ns Tsetup (TFF): 3ns Thold (TFF): 1ns
Tprop (DFF): 9ns Tsetup (DFF): 4ns Thold (DFF): 1ns
Tprop (JKFF): 10ns Tsetup (JKFF): 3ns Thold (JKFF): 1ns
Tprop (SRFF): 8ns Tsetup (SRFF): 4ns Thold (SRFF): 1ns

Note: You must show the path you used to get your answer in order to get credit.



[9] Given the following table, draw the Karnaugh maps for Y1', Y2' and Z in terms of X, Y1 and Y2, and then write **minimum** boolean equations for each. Do not worry about the fact that the machine might get stuck in a particular state and be unable to get out - just solve the problem as presented.

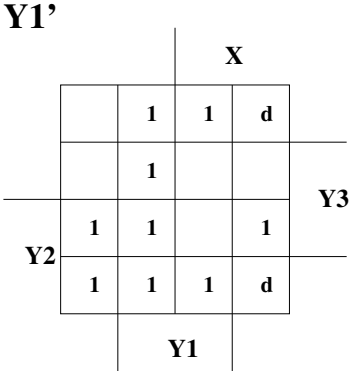
Present State (Y1 Y2)	Next State X=0 (Y1' Y2')	Next State X=1 (Y1' Y2')	Output X=0	Output X=1
01	11	10	1	0
10	01	11	0	1
11	11	01	0	0



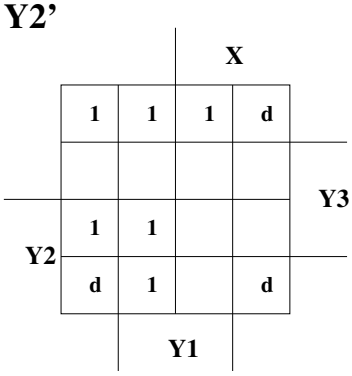
Desired Transition Y -> Y'	SR FF S R	JK FF J K	T FF T	D FF D
0 -> 0	0 d	0 d	0	0
0 -> 1	1 0	1 d	1	1
1 -> 0	0 1	d 1	1	0
1 -> 1	d 0	d 0	0	1

[15] Given the following Karnaugh maps, implement the sequential machine using a JK FF for Y1, an SR FF for Y2, and a T FF for Y3. You do not need to draw the gates, but you do need to write down the **minimized** input equations for each of the inputs of each of the Flip Flops in the circuit.

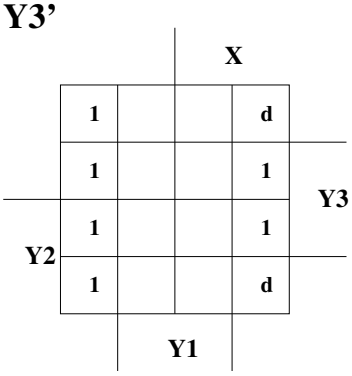
Y1'

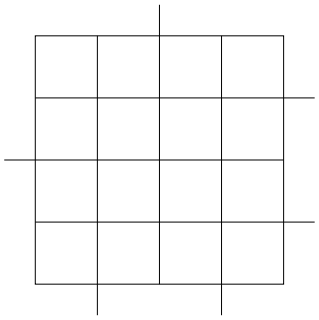


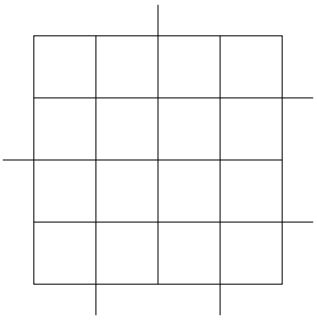
Y2'

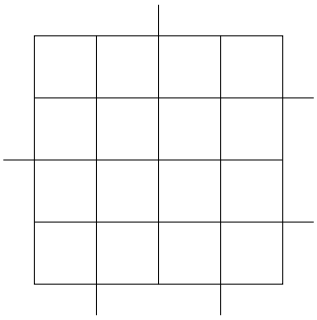


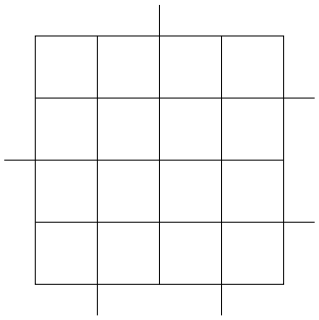
Y3'

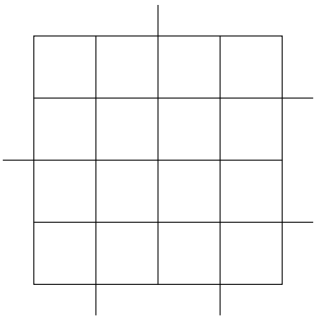


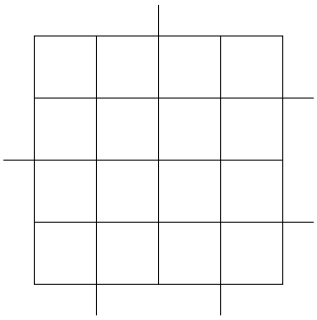












Desired Transition Y -> Y'	SR FF S R	JK FF J K	T FF T	D FF D
0 -> 0	0 d	0 d	0	0
0 -> 1	1 0	1 d	1	1
1 -> 0	0 1	d 1	1	0
1 -> 1	d 0	d 0	0	1

[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.

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

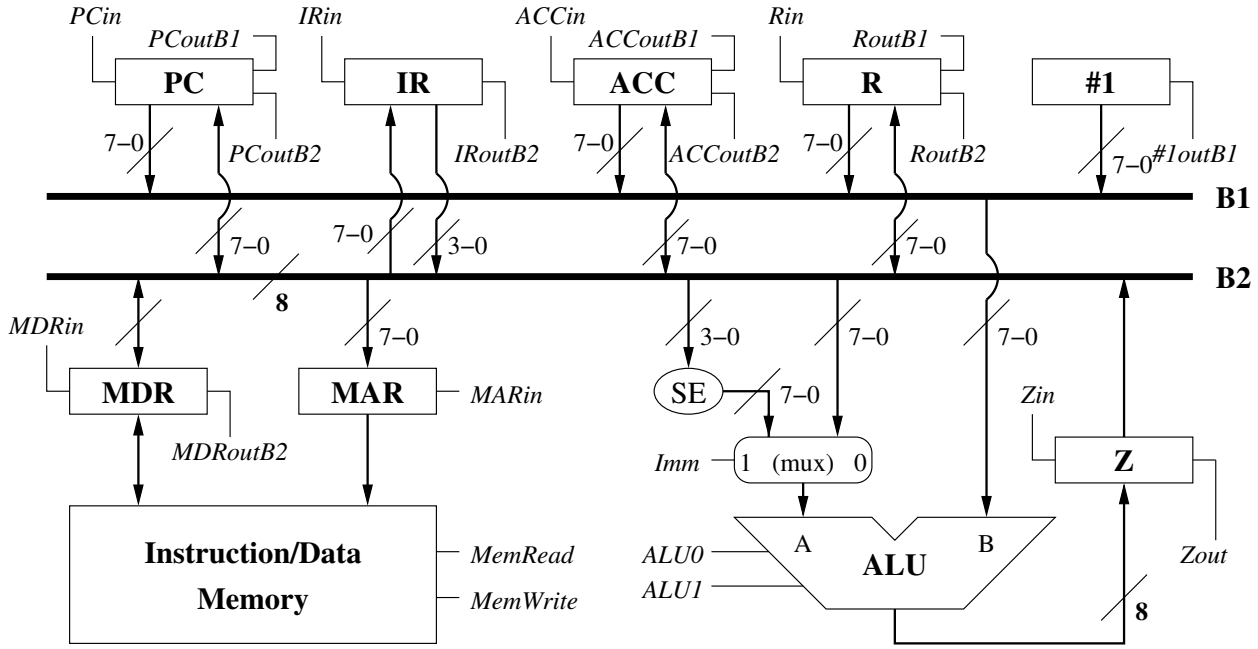
Here are a few of the instructions that have been defined:

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																						

[3] Given the following 18-bit address and a 256-byte Direct Mapped cache with a linesize=2, show how an address is partitioned/interpreted by the cache.

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

[4] Assuming a 18-bit address and a 128-byte 4-way Set Associative cache with a linesize=8, show how an address is partitioned/interpreted by the cache.

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

[2] Assuming a 18-bit address and a 128-byte Fully Associative cache with a linesize=4, show how an address is partitioned/interpreted by the cache.

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

[5] Given a 1Mbyte physical memory, a 24 bit Virtual address, and a page size of 2K bytes,

How many entries are there in the page table?

How wide is each entry?

Is there a problem with this configuration? If so, how can you fix it?

[6] Given a 2 Megabyte physical memory, a 32 bit Virtual address, and a page size of 4K bytes,

How many entries are there in the page table?

How wide is each entry?

Is there a problem with this configuration? If so, how can you fix it?

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?

[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.

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

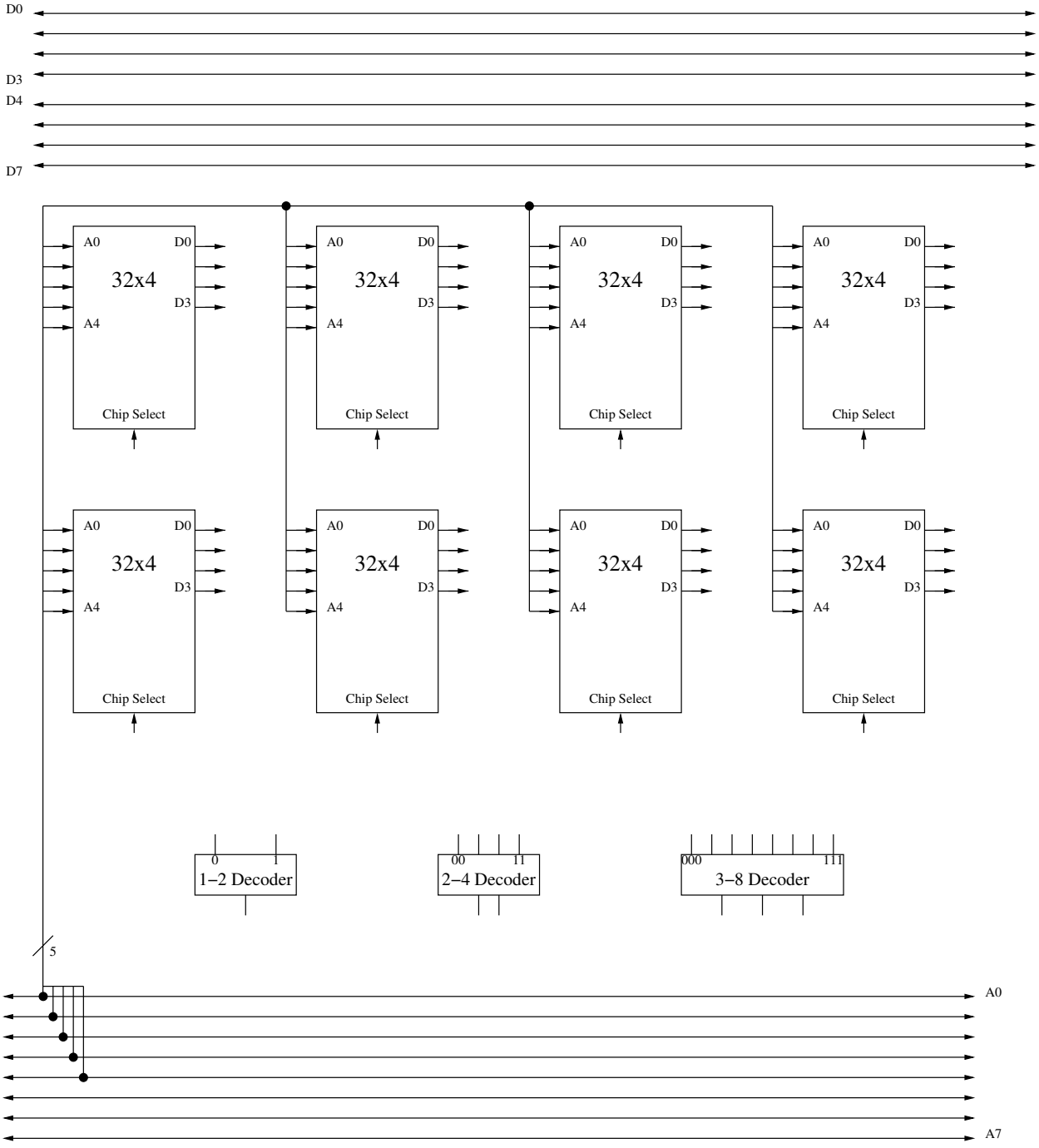
[9] For each of the following convert the virtual address into a physical address (if possible) and write down the value of the memory location corresponding to the address. If it is not possible to do so, explain why.

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).

[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.



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

2^4 = 16
2^8 = 256
2^10 = 1K
2^20 = 1M
2^30 = 1G

0x1 = 0001
0x2 = 0010
0x4 = 0100
0x8 = 1000