Solutions to Homework 5, including Study Questions

4.20 "bub" indicates bubble

addi $s0, $s1, 5	IF	ID	EXE	MEM	WB							
NOP		bub										
NOP			bub									
NOP				bub								
add $s2, $s0, $s1					IF	ID	EXE	MEM	WB			
addi $s3, $s0, 15						IF	ID	EXE	MEM	WB		
NOP							bub					
NOP								bub				
add $s4, $s2, $s1									IF	ID	etc.	

Note: We can reduce both groups of NOPs by one if we assume the WB stage can be split into separate write/read portions within one cycle

4.25.1 Note: "bub" indicates bubble

ld $s0, 0($s3)	IF	ID	EXE	MEM	WB											
ld $s1, 8($s3)		IF	ID	EXE	MEM	WB										
NOP			bub	:(	:(	:(	:(									
add $s2, $s0, $s1				IF	ID	EXE	MEM:(	WB								
addi $s3, $s3, -16					IF	ID	EXE	MEM:(	WB							
bnez $s2, LOOP						IF	ID	EXE	MEM:(	WB:(						
ld $s0, 0($s3)							IF	ID	EXE	MEM	WB					
ld $s1, 8($s3)								IF	ID	EXE	MEM	WB				
NOP									bub	:(	:(	:(	:(			
add $s2, $S0, $s1										IF	ID	EXE	MEM:(	WB		
addi $s3, $s3, -16											IF	ID	EXE	MEM:(	WB	
bnez $s2, LOOP												IF	ID	EXE	MEM:(	WB:

4.25.2 ":(" above indicates a stage with no useful work

There are NO cycles in which every stage is doing useful work. Yikes.

4.27.1 The following assumes that each stage takes the entire cycle to complete. If the stages can be split in half, then fewer NOPs could be used. For example, if WB takes the first half of a cycle and ID uses the second half, then those two stages can be done at the same time.

Note: "bub" indicates bubble

add $s3, $s1, $s0	IF	ID	EXE	MEM	WB									
NOP		bub												
NOP			bub											
NOP				bub										
lw $s2, 4($s3)					IF	ID	EXE	MEM	WB					
lw $s1, 0($s4)						IF	ID	EXE	MEM	WB				
NOP							bub							
NOP								bub						
or $s2, $s3, $s2									IF	ID	EXE	MEM	WB	
sw $s2, 0($s3)										IF	ID	EXE	MEM	WB

4.27.2

add $s3, $s1, $s0	IF	ID	EXE	MEM	WB					
lw $s1, 0($s4)		IF	ID	EXE	MEM	WB				
NOP			bub							
NOP				bub						
lw $s2, 4($s3)					IF	ID	EXE	MEM	WB	

By changing the position of lw $s1, you can reduce the initial NOPs to two, as shown above. However, the same kind of dependency emerges later with $s2 which will necessitate three NOPs, so we will still have a total of five. Thus, it can't be done without forwarding.

But, **with** forwarding, we can do this:

1.	add $s3, $s1, $s0	IF	ID	EXE	MEM	WB				
2.	lw $s2, 4($s3)		IF	ID	EXE	MEM	WB			
3.	lw $s1, 0($s4)			IF	ID	EXE	MEM	WB		
4.	or $s2, $s3, $s2				IF	ID	EXE	MEM	WB	
5.	sw $s2, 0($s3)					IF	ID	EXE	MEM	WB

From instruction 1, forward $s3 from EXE to EXE of instruction 2.

Instruction 3 has no dependencies.

From instruction 1, forward $s3 from EXE to EXE of instruction 4. Also foward the result of instruction 2's MEM stage to EXE of instruction 4. Thus, all of the dependencies of the `or` instruction are satisfied.

Finally, forward the result of instruction 4's EXE to instruction 5's MEM stage.

It makes more sense if you draw out the pipeline.

5.5.1 Offset = 5 bits = $2^5 = 32$ bytes. Since there are 4 bytes/word, $32/4 = 8$ words.

5.5.2 $2^5 = 32$ entries.

5.5.4

Ref.	Tag	Index	Offset	Hit/Miss	Replace		Index	Cache
0	0-00	00000	00000	M	–		0	[0, 4, 16], [1024], [30], [3100]
4	0-00	00000	00100	H	N		1	
16	0-00	00000	10000	H	N		2	
132	0-00	00100	00100	M	–		3	
232	0-00	00111	01000	M	–		4	[132, 140], [2180]
160	0-00	00101	00000	M	–		5	[160, 180]
1024	0-01	00000	00000	M	Y		6	
30	0-00	00000	11110	M	Y		7	[232]
140	0-00	00100	01100	H	N		8	
3100	0-11	00000	11100	M	Y		9	
180	0-00	00101	10100	H	N		:	
2180	0-10	00100	00100	M	Y		31	

- Using a byte address means that individual bytes can be accessed.
- The entire address is 32 bits.
- The leftmost 22 bits are the tag; the table shows the first and last two bits of this field.
- The next 5 bits are the index; this references the cache block. Since this is 5 bits, there are $2^5 = 32$ blocks in the cache.
- The Hit/Miss column indicates that a cache block has been indexed again.
- N in the Replace column indicates if the address is referencing the same block already in the cache; a Y indicates that the block must be replaced. The first time a block is indexed, it is shown as a –.
- The last two columns show the cache itself and the references that were put into each block, in left-to-right order. The brackets indicate bytes that are within the same 8-word block.

5.5.5 The hit ratio is $\frac{4}{12} = 0.3\bar{3}$, or 33%.

Additional problems:

1.

Dependency	Register	Instructions
RAW	R1	1 – 2
RAW	R1	1 – 3
RAW	R2	2 – 3
WAR	R1	2 – 3
WAR	R2	1 – 2
WAW	R1	1 – 3

2.

Ref	Tag	Index	H/M
3	0000	0011	M
180	1011	0100	M
43	0010	1011	M
3	0000	0011	H
191	1011	1111	M
35	0010	0011	M
3	0000	0011	M
3	0000	0011	H
35	0010	0011	M
190	1011	1110	M
191	1011	1111	H
174	1010	1110	M
190	1011	1110	M
35	0010	0011	H
35	0010	0011	H
190	1011	1110	H
191	1011	1111	H
3	0000	0011	M
180	1011	0100	H

Additional text problems:

5.16.1

These are generally too long for the final exam, but you should have a good idea how to access the page table and what the TLB does. The following explains what happens with the first virtual page access.

Each page has size 4 KiB = 4096 bytes. The address stream shows byte addresses. Thus, we need 12 bits for the byte portion of the virtual address. In order to store the largest virtual page address (49225), we need 16 bits. Subtracting the 12 bits for the byte portion leaves us with a 4-bit page table index, which is enough for the 12 entries in the given page table.

The first address in the stream is 4669. This is a virtual address of 0001001000111101. The virtual page number (the first 4 bits) is 0001. We simultaneously look at the Tag field in the TLB (because it is fully associate, so the page can go in any of the four locations) and the second row of the page table. It is not in the TLB, so it is a TLB miss. The entry in the page table indicates the page is not in RAM (valid bit is 0), so we have to go to disk and we have a page fault. According to the instructions, we will get the next largest page number, or page 13. The page table gets updated to "valid" and physical page 13; the TLB also gets updated, changing the 0 in the valid row to 1 and putting 13 there as the physical page number as well.

5.16.2

The larger page size increases the TLB hit rate but can lead to higher fragmentation (less of each page is actually used).

5.16.3

Concentrate on n-way set associative caches rather than on the TLB.