Ex. 1

Index	Valid	PPN or on Disk
0	1	5
1	1	0
2	1	1
3	1	6
4	1	9
5	1	11
6	0	Disk
7	1	4
8	0	Disk
9	0	Disk
10	1	3
11	1	12

Valid	VPN	PPN	LRU
1	1	0	3
1	7	4	1
1	3	6	2
1	2	1	4

Ex. 2

Since the number of frames is equal to the size of the memory divided by the page size, increasing the page size will proportionately decrease the number of frames.

Having fewer frames will tend to increase the number of page faults because of the lower freedom in replacement choice. Imagine a system with four frames with the reference history of 0, 4, 3, 1. On a page fault, LRU would victimize frame 0. With a doubling of page size the reference history becomes 0a, 1b, 1a, 0b; so the LRU victim would be 1 (corresponding to small page frames 3 and 4) when one would prefer to victimize 0a (the first half of 0).

Large pages will also waste more space with internal fragmentation. If a typical process has three sections (text, heap, stack), on average about half a page per section is unused so 1.5 pages worth of memory are unused per process. If page size is doubled, this waste is doubled.

On the other hand, using larger pages will draw in more memory per fault, so the number of faults may decrease if there is limited contention and/or reasonably high spatial locality at the scale of page size (e.g., references to the high half of a double-size page occurring near in time to references to the low half would make replacement of a double-size page a close approximation of the replacement for the two smaller pages of which it is composed). (If memory is abundant, first reference [a.k.a. compulsory] page faults will tend to dominate, so larger pages will reduce the number of page faults.) Of course, the OS can use prefetching to accomplish the same reduction in faults with the benefit of being able to throttle prefetching under heavy memory pressure or poor prefetch behavior while avoiding the above mentioned disadvantages of large pages.

Of course, larger pages do reduce the number of TLB misses (sometimes called minor page faults), and an OS can support multiple page sizes and aggregate smaller pages to form larger pages (for reducing TLB misses) and deaggregate larger pages to from smaller pages (to reduce the volume of memory swapped and to reduce the above negative effects of larger pages).

Ex. 3

Virtual address bits	Physical address bits	Page Size	VPN bits	PPN bits
32	32	16KB	**18**	**18**
32	26	**8KB**	**19**	13
36	32	**32KB**	21	17
40	**36**	32KB	25	21
64	40	**64KB**	48	**25**

To solve this task we can use following formula:
VA=VPN + offset, PA=PPN + offset, offset = $\log_2$(Page Size)

Ex. 4

2 GB RAM => PA = 31 bit
Page Size = 2KB
VA = 36 bit
Flags = 12 bit
PPN, VPN = ?
TableSize = ?

Offset = 11 bit
PPN = 20
VPN = 25

VA = [flags + PPN]
TS = VA_{amount} * VA_{size} = > 2^5 * 2^25 = 2^30 bit = 128 MB

Table Size = 128 MB

Ex. 5

N = 8
C = 4 KB
b = 16 Byte
Write back | LRU = 2 bits

B = C/b = 2^12 / 2^4 = 2^8
S = B/N = 2^5
set bits = 5 bits

word = 4 Byte
block offset = log2(b/word) = 2 bit

Address = [tag (23) | set(5) | block offset(2) | byte offset(2)]
1 + 153 * 8 = 1225 bit
Full Size = 1225 * 32 / 8 = 4900 byte

Ex. 6

b = 512 Byte
C = 512 KB
1. D.M.C = ? // S = B = 1024, N = 1
2. F.A.C = ? // S = 1, N = B = 1024
3. 4-way = ? // S = B/4 = 256, N = 4
B = ?

B = C/b => B = 1024
1) [tag(12) |set(1) | block offset (9)]
2) [tag(22) | block offset(9)]
3) [tag(15) | set(8) | offset(9)]

Ex. 7

1 – Total cycles
2- CPI

1) 2*4 + 1*4 + 10*(4 + 3 + 4 + 4 + 3) + 4 + 3 = 12 + 180 + 7 = 199
2) R-type = 1 + 10 + 10 + 1 = 22;
   add = 12
   beq = 11
   j = 10

CPI = (22/55 + 12/55)*4 + ((10+11)/55)*3 = 3.62

Ex. 8

1) Multi-Cycle

   T_{mc} = 5 ns //The longest operation

   T_{mc} = 4*4 + 3 = 19 t //Tacts in the program

   $S_{mc} = t_{mc} * T_{mc}$ = 5 * 19 = 95ns //Summary time

2) Single-Cycle

   $t_{sc} = t_{pcq\_PC} + t_{mem} + \max(t_{RFread}, t_{sext} + t_{mux}) + t_{ALU} + t_{mem} + t_{mux} + t_{RFsetup}$

   t_{sc} = 0ns + 5ns + 2ns + 3ns + 5ns + 0ns + 2ns = 17ns

   T_{sc} = 5t

   S_{sc} = 17*5 = 85ns