Computer Science 605.411

Module 10 Example Set 1

1. A virtual memory system employs three levels of page map tables with 50-bit virtual addresses and 32-bit physical addresses. The page size is 16384 bytes. The size of each page map table is the same as the page size. PTEs (page table entries) within all tables are 4 bytes wide.
   a) (10) How many bits would be required for each virtual page number?
      Since the page size is 16384 bytes, 14 bits are required for the page offset. Hence 50-14 = 36 bits are required for the virtual page number.
   b) (10) Identify the numbers (counting from 0) of all page tables and page table entries that would be accessed in translating the virtual address 0x48D159E26AF0.
      Since there are 3 levels of page tables, the 36-bit virtual page number field would be subdivided into three 12-bit fields. The virtual address 0x48D159E26AF0 would be partitioned as 000100100011 010001010110 011110001001 101010111110000.
      Hence the page tables and entries that would be accessed are:
      000100100011 = 291 in the level 0 page table (the top level table)
      010001010110 = decimal 1110 in the level 1 page table
      011110001001 = decimal 1929 in the level 2 page table

2. A system employs a single inverted page map table in which each page table entry contains four items: a valid bit, a dirty bit, a 2-bit ID and the information needed to complete the page to frame translation. If this system employs 48-bit virtual addresses, 134217728 bytes of main memory, and 4096-byte pages,
   a) (5) what is the minimum physical address size in bits?
      A minimum of 27 bits would be required to address 134217728 bytes of main memory. $Log_2(134217728) = 27$.
   b) (5) what is the minimum size in bits of each entry in the inverted page map table?
      The number of bits in the page number is 48-12 = 36.
      Each entry in the inverted page map table would contain a 36-bit page number, a valid bit, a dirty bit, and a 2-bit ID for a total of 40 bits.
   c) (5) how many entries would the inverted page map table contain if the memory size was changed so that the physical address was 30 bits wide?
      A 30-bit physical address would correspond to 2^{30} bytes of memory, so there would be $2^{30}/2^{12} = 2^{18}$ frames. Hence there would be 262144 entries in the inverted page table.

3. Two independent non-communicating processes are running concurrently on a virtual memory system with a single unified cache. The processes do not share any physical memory frames. The first process reads from some virtual address followed by the second process reading from a different virtual address and each read causes a cache hit.

a) (10) Could these two references cause that same line within the cache to be accessed if the cache is referenced using virtual addresses? Explain your answer.

Yes. With a virtual memory system two processes may refer to the same virtual address but each would have its virtual address mapped to a different physical address. Even if the virtual addresses are not the they could contain the same tag and set or line number fields, so they could both map to the same cache line. This would be the case if the two addresses only differed in the offset field. Additional information such as the process ID would have to appear in the cache to avoid this problem if virtual addresses are used to access the cache.

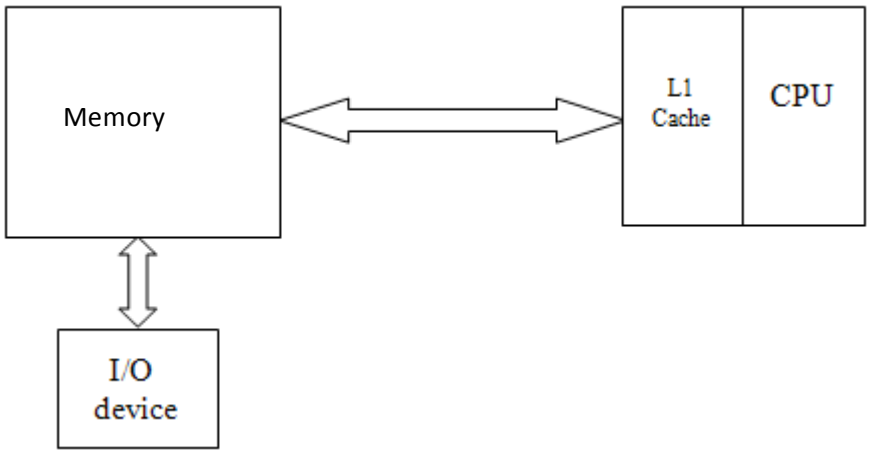
b) (10) Would your answer be the same if the cache had been referenced using physical addresses rather than virtual addresses? Explain.

No. The virtual addresses used by each process would have been translated into different and distinct physical addresses using separate page map tables. Since the two processes are independent and non-communicating, the OS would insure that the frames assigned to one process never match those assigned to the other. Therefore the two references could not map to the same cache line if physical addresses are used to access the cache.

4. (15) An executing program spends 65% of its time on disk I/O. If the disk system is replaced with an improved disk system that provides 50% greater throughput (i.e. the new disk system can perform the same operations as before but 50% faster), what overall speedup would be achieved for the program if the only change is to use the new and improved disk system?

If the disk system has 50% greater throughput, it will provide 1.5 times the throughput of the original system.
Based on Amdahl's law, if T is the total CPU time before the improvement then the time after the improvement is $T(0.35 + 0.65/1.5)$.
Hence the speedup = $T_{before}/T_{after} = T/\ T(0.35 + 0.65/1.5) = 1.28$

5. Suppose that our MIPS system employed a write allocate policy for its unified copy-back L1 cache. The system contains an I/O device with an integrated I/O controller that directly accesses memory without involving the cache as shown in the diagram below:



The following instructions are executed by the CPU on this system to store the 32-bit pattern 0xABCDEF12 into memory at the address 0x07E48080:

```
lui     $t0,0x07E4
ori     $t0,$t0,0x8080
lui     $t1,0xABCD
ori     $t1,$t1,0xEF12
sw      $t1,0($t0)
```

The memory employs little-endian storage order.
After these instructions have executed, the I/O device then writes a series of 4 bytes corresponding to 0x12, 0x34, 0x56, 0x78, in that order, into memory starting at address 0x07E48082. After the I/O device completes its writes, the CPU executes the following three instructions:

```
lui     $t0,0x07E4
ori     $t0,$t0,0x8080
lw      $t2,0($t0)
```

(10) a) Use 8 hex digits to show the 32-bit pattern loaded into $t2 by these instructions.
Due to the write-allocate policy, the write performed by the CPU will cause the block containing address 0x07E48080 to be updated and brought into the cache. Since the I/O write goes directly to memory bypassing the cache, the cache will not be updated to reflect the write performed by the I/O device. When the CPU again reads from address 0x07E48080, it will obtain from cache the same pattern that it had previously written. Therefore the contents of $t2 = 0xABCDEF12.

(10) b) If the memory had used big-endian storage order rather than little-endian storage order what would be the contents of $t2?

Using big-endian storage would affect the order of the bytes within a word in memory, however, when the word is read from memory or from cache the bytes are always placed into the register from high byte on the left to low byte on the right. Due to the cache hit, the CPU obtains the data from cache and $t2 would still contain 0xABCDEF12 the original value written by the CPU.

6. (10) The instruction sequence below runs on our non-pipelined multi-cycle datapath and employs polling to input a stream of characters coming from the keyboard attached to a USB interface that has a bandwidth of 115200 bytes per second (including any required overhead, such as parity, etc.). The CPU clock rate is a relatively slow 100 MHz. The hardware sets the ready bit within the control/status register each time a new character is received; the ready bit is automatically cleared to 0 when the input character is read by the instruction sequence. Delayed branching is not used since this is a non-pipelined system. No cache is used with this system.

```
        lui   $t3,0xFFFF      # point $t3 to the control/status register
loop:   lw    $t1,0($t3)      # read and check the ready bit
        andi  $t1,$t1,1       # mask off all but the ready bit
        beq   $t1,$0,loop     # if not set, continue checking
        nop
        lw    $t0,4($t3)      # read the input character
        beq   $0,$0,loop      # poll for the next character
```

While this code is running, a typist enters a stream of characters at the rate of 320 keystrokes per minute. On average, how many times will the ready bit be checked by this code between consecutive characters entered while the typist is typing? Assume that the beq instruction requires 3 cycles, the lw instruction requires 5 cycles and all other instructions require 4 cycles each.

The hardware is capable of transmitting 115200 characters per second, however the limiting factor is the typing speed. The time interval between keystrokes would be 60 seconds/320 = 0.1875 seconds. The cycle time is 1/100MHz = 10 ns. The time to execute the loop would be the sum of the execution time for the lw, andi & beq instructions = 5+4+3 = 12 cycles . Twelve cycles corresponds to 120ns. Hence the number of times the ready bit is checked between keystrokes = 0.1875/1.2e-7 = 1562500. Each time that the ready bit is found to be set, the final 3 instructions will also be executed. These account for 4+5+3 = 12 cycles or 120 ns. Hence after the first character arrives, the time interval during which the loop executes before the next character arrives is 187500000 – 120 = 187499880 ns. Therefore the loop will be executed 187499880/120 = 1562499 times between consecutive characters.

Module 10 Example Set 2

1. An instruction cache (I-cache) has a storage capacity of 524288 bytes and employs 128-byte cache lines. The memory with which the cache is used is 209715200 bytes in size.

   a) How many instructions can each set hold at one time if the I-cache has a 2-way set associative organization?

   Each set would contain two 128-byte lines. Each line holds 128/4 = 32 instructions, so each set can hold 2*32 = 64 instructions.

   b) How many sets are in the cache?

   There are two lines per set, so each set is 2*128 = 256 bytes in size.
   The number of set is the total cache size divided by the set size = 524288/256 = 2048.

   c) How many blocks are there in the memory if it is used with this I-cache?

   Since the memory block size is always the same as the cache line size (128 bytes in this case), the number of blocks is just the memory size divided by the block or line size: 209715200/128 = 1638400 blocks.

   d) To which set would the instruction at address 0x4008400C map if the I-cache were organized as a 4-way set associative cache?

   In this case, each set would contain 4 lines and is therefore 4*128 = 512 bytes in size. So the number of sets is 524288/512 = 1024, Therefore the set number field requires 10 bits. $Log_2(1024) = 10$. The width of the offset is 7 bits $[log_2(128)]$ and the following address format applies:

Tag		Set number		Offset	
31	17	16	7	6	0

   0x4008400C = binary 01000000000100 0010000000 0001100
   This address maps to set 0010000000 = 0x080 = 128

   e) To which memory block would this address correspond?

   The memory blocks are the same size as the cache lines (128 bytes), and thus have the same size offset field (i.e. 7 bits). The bits to the left of the offset field would give the block number. This is the same number obtained by the integer division of the address/128.
   The block number is binary 01000000000010000010000000 = 0x801080 = 8392832

f) How many different memory blocks could potentially map to set 10 within the 4-way set associative cache?

Any memory block whose block number has bits 16 through 7 equal to 0000001010 would map to set 10. Since the memory is 209715200 bytes in size (=0xC800000), so only the low 28 bits of the address will ever be none-zero. That is, at most 28 bits are needed to specify any address within the memory. So the upper 28-10-7 = 11 bits would determine how many different blocks map to an individual set. The value in the upper 11 bits range from 0 to 11000111111 = decimal1599. So 1600 different blocks map to set 10 within the 4-way set associative cache.

2. A system has a unified cache with a 10ns access time, and a memory with a 120 ns access time.

a) With the cache operating in look-through mode, the effective access time for reads is 22ns. What would be the corresponding hit ratio for reads?

With look-through mode, the effective access time for reads is
T = 22 = 10 + (1-h)*120 = 130 - 120h where h is the hit ratio.
Hence h= 108/120 = 0.9 or 90 %

b) If the system has a read hit ratio of 85% with the cache operating in look-aside mode, what is the corresponding effective access time for reads?

With look-aside mode, the effective access time for reads is
T= h*10 + (1-h)120 = 0.85*10 + 0.15*120 = 26.5 ns.

3. On a system that employs multiprogramming with virtual memory, how is one process prevented from accessing or overwriting the pages that belong to another process? The OS would insure that the frames assigned to one process never match those assigned to the other. This is done by making sure that the page map table for the two processes contain distinct and different page to frame mappings.

4. What difference does it make whether a system, containing a cache and employing virtual memory, uses the 32-bit physical or 32-bit virtual address to access the cache when running two different processes concurrently?

None of the physical addresses generated by one process would match those generated by another process, so the cache would only be accessed using different and distinct physical addresses.

With virtual addresses, two different processes could reference the same virtual address even though their respective page map tables would translate the common virtual address to different physical addresses. Therefore the cache could be referenced with the same virtual address by two different processes.

5. A system implements a paged virtual address space for each process. The page size is 1024 bytes. The maximum size for the virtual address space is 128×2^{20} bytes and the physical address space size is 2×2^{20} bytes. The page table for the running process includes the following five valid entries (where the => notation indicates that a virtual page maps to a given frame (i.e., the page is located in that frame):

Virtual page 2 => frame 4
Virtual page 1 => frame 2
Virtual page 0 => frame 1
Virtual page 4 => frame 9
Virtual page 3 => frame 16

a) What is the minimum number of bits required to cover the virtual address space? 27 bits are required to cover a 128 MB address range.
$\text{Log}_2(128\times 2^{20}) = 7+20=27$ bits.

b) What is the minimum number of bits required to cover the physical address space?
21 bits are required to cover a 2 MB address range.
$\text{Log}_2(2\times 2^{20}) = 1+20=21$ bits.

c) What is the maximum number of entries in a single-level page map table? The maximum number of PTEs would be the same as the maximum number of pages = 128×2^{20} /1024 = 128 * 1024 = 131072.

d) To which physical address will the virtual address decimal 1524 translate? Decimal 1524 = 0x5F4. Ten bits are required for the page offset within the 1024-byte pages. The leftmost 27-10 = 17 bits within the virtual address gives the page number. Hence the page number is 1. Page 1 maps to frame 2 therefore the corresponding physical address is binary 100111110100 = 0x9F4 = decimal 2548.

e) Which virtual address will translate to physical address decimal 17880? Decimal 17880 = 0x45D8, so the frame number = binary 010001 = 17 (the leftmost 11 bits of the physical address). None of the valid PTEs correspond to frame 17, so no virtual address would translate to this physical address until some page has been brought into memory and assigned to frame 17.

6. A system with $4 * 2^{30}$ bytes (4 GB) of memory employs a unified fully associative cache with a line size of 256 bytes and a total of 8192 cache lines. Currently line 768 within the cache is valid and has a tag = 0x23E450. Which memory block currently occupies line 768 within the cache?
The memory block size is always the same as the cache line size (256 bytes in this case). The offset field requires 8 bits to cover the range 0 through 255. Hence the tag is made up of the bits to the left of this 8-bit offset field. Given any memory address for this system, the block number would be the bits to the left of the offset field. Hence the block number and the tag are the same. So line 768 contains memory block 0x23E450.

7. a) How many blocks would a system that contains $4 * 2^{30}$ bytes (4 GB) of memory contain if the block size is 4096 bytes?
This would just be the total memory size / blocksize = $2^{32} / 2^{12} = 2^{20}$.
b) To which memory block would the address 0x209788CE correspond?
Block number = (address / block size) = 0x209788CE / 4096 = 0x20978.
Note that the same result can be obtained by truncating the low 12 bits of the address.

8. Set 10 within a 4-way set associative cache is currently full and the pseudo-LRU bits for set 10 were most recently updated from 110 to 101.

a) What line within set 10 was most recently accessed?
If the pseudo-LRU bits go from 110 to 101, then B1 went from 1 to 0, and B0 went from 0 to 1. These are the updates that would be made if way 1 within the set is accessed as shown in the table below from module 10:

Way accessed	Effect on LRU bits
0	1->B1, 1->B0
1	0->B1, 1->B0
2	1->B2, 0->B0
3	0->B2, 0->B0

b) If the very next memory access corresponds to a hit within way 3 of set 10, to what 3-bit pattern should the pseudo-LRU bits be changed?
Since way3 is accessed, B2 should go to 0 and B0 should go to 0. Hence the pseudo-LRU bits change from 101 to 000.

c) If the next memory access results in a miss in set 10, what action should be taken?
In response to the miss, a line within the full set 10 will have to be replaced. Since the pseudo-LRU bits are 000, way0 should be replaced as shown in the table below:

B2 B1 B0	Way to replace
000	0
001	2
010	1
011	2
100	0
101	3
110	1
111	3

Replacing way0 is also a reference, so the pseudo-LRU bits should be updated from 000 to 011 based on the table in part a).

9. A matrix of 32-bit integers, x[512][32] resides at memory address 3EA4000 on a machine with a virtual memory system that employs 8192-byte pages. The matrix contains 512 rows and 32 columns. Assume that the amount of physical memory available to contain pages from the matrix is 32768 bytes. Note that in the C language, matrices are stored in row major order (i.e. one row after another). Recall that memory is byte addressable and its size is measured in units of bytes not bits.

a) How many frames are required to hold the complete matrix?

There are 512 rows. Each row contains 32 elements. Each element is 4 bytes in size. Hence the total size of the matrix is 512*32*4 = 65536 bytes. Frames and pages have the same size (8192 bytes in this case). Hence the matrix corresponds to 65536/8192 = 8 frames.

b) How many different pages are referenced by the following code sequence:

```
for (j = 0; j < 3; j++) {
        for (k = 0; k < 35; k++)    x[k][j]  = x[k][j] + 1;
}
```

One row of the matrix corresponds to 32 four-byte elements or 128 bytes. Hence each page can hold 8192/128 = 64 rows. The code references columns 0, 1 and 2 within each of the first 35 rows. The first 64 rows of the matrix are contained in a single page, so the code only references a single page.

Module 10 Example Set 3

1. How does a regular or forward page map table differ from an inverse page map table?
A forward page map table (PMT) contains an entry (PTE) for each logical or virtual page while an inverted page map table contains an entry for each of the frames (i.e., physical pages). The number of frames = (physical memory size) / (frame size).

2. Given a specific physical address, how can the corresponding memory block number be computed?
For a given frame size N, the number of bits in the offset field is F = $\log_2 N$. The bits within the physical address to the left of this offset field contain the memory block number. This is equivalent to the integer quotient produced when the address is divided by the frame size.

3. What distinguishes a physical memory block from a logical memory page?
Logical pages are the same size as physical memory blocks, but the logical pages exist within the logical address space which may exceed the size of the physical address space. Physical memory blocks correspond to the actual hardware memory available in the system. A given logical page may be mapped into any available memory block. Adjacent logical pages do not have to reside within adjacent physical memory blocks. On a virtual memory system, logical pages are called virtual pages (or just pages) while physical memory blocks are called frames.

4. For a direct mapped cache, how would the cache line number be computed for a particular physical memory ?
Memory is thought of as being subdivided some number of blocks. Each block has the same size as a cache line and fits exactly into a single cache line. Given the memory block number B, the line to which the block maps = B % L (i.e., B modulo L) where L is the number of lines in the cache.

5. A logical address can be viewed as containing three fields for a direct mapped cache system. What determines the number of bits contained within each logical address?
The rightmost field is the offset within the selected line and requires a number of bits = $\log_2$(line size). The next field to the left contains the line number and requires a number of bits = $\log_2$(lines in cache). The next field to the left contains all of the remaining bits from the address and indicates the tag for the memory block that maps to the cache line.

6. What is meant by a “byte addressable” menory system?
This is one in which memory access instructions can reference items as small as individual bytes within the memory system. For a memory system of size 2^N, the address of the first byte is 0 and the address of the final byte is $2^N - 1$.

7. For a big-endian memory system that employs 4-byte words, to which byte within the word does the word address correspond?
For a big-endian system, the word address corresponds to the most significant (i.e., left-most) byte within the 4-byte word. On a little-endian system, the address would corrspond to the least significant (i.e., right-most) byte within the 4-byte word.

8. When a "load byte" or a "load half-word" instruction is executed on our MIPS system, the load byte places one byte into the low 8 bits of the destination register but the load half-word places two bytes into the low 16 bits within the destination register. What happens to the remaining bytes within the destination register?
The remaining bits within the register are filled with a copy of the MSB in the low byte if the instruction is either lb or lh, but if the instruction is either lbu (load byte unsigned) or lhu (load half-word unsigned) the remaining bits within the register are filled with 0s.

9. What is meant by an un-aligned memory reference?
This refers to an attempt to read or write a multi-byte item at a memory address that is not a multiple of the size of the multi-byte item. For example, an attempt to access a word at an address that is not a multiple of 4, or to access a half-word at an address that is not a multiple of 2.

10. a) A memory system contains 64 memory chips each of which has a width of 32 and a depth of 8388608. What is total storage capacity in bytes of this memory system?
Since each chip is 32 bits wide, each memory cell contains 4 bytes. The depth indicate how many cells are in each chip, hence each chip contains 4*8388608 = 33554432 bytes. The total number of bytes for the system = 64 * 33554432 = 2147483648 bytes (i.e., 2 GB).
b) By how much would the addresses of adjacent cells within each chip differ?
Since each cell is 4 bytes wide, the addresses of adjacent cells would differ by 4.

c) Assuming the system uses a 32-bit data bus, how many chips would have to be accessed to transfer one word?
If the address used is properly aligned (i.e., is a multiple of 4), then only one chip would have to be accessed. Using an address that is not a multiple of 4 would require accessing two chips since the bytes that comprise the word would span two chips.

d) Assuming the system uses 32-bit addresses and enforce memory alignment, how would the bits within each address be used to identify which cell to access?
Requiring that the address be a multiple of 4 means that the low 2 bits of the address would always be 00. Hence the low 2 bits need not be used. Since $\log_2(8388608) = 23$, the next 23 bits to the left would identify which cell in the selected chip to access. The next 6 bits to the left would select a particular chip (since there are $2^6 = 64$ chips). The 32-bit (i.e., 4-byte) cell contents would be transferred into the MDR (memory data register) and copied from the MDR into the destination register. Hence the address format would be:

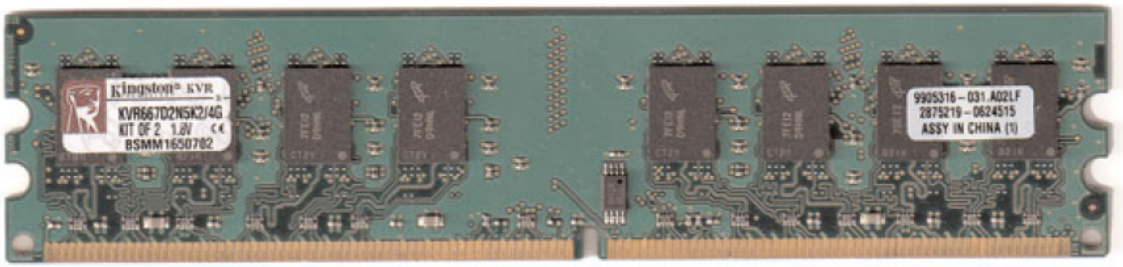
Bit 31 = 0 since size is only 2 GB	Bits 30 to 25 = chip-select	Bits 24 to 2 = cell# within chip	Bits 1 to 0 = byte select within MDR

e) Why are unaligned memory references a problem or disadvantage for this system?
For a read the contents of the 32-bit cell would be read from the chip and transferred into the MDR (memory data register) and the low 2 bits of the address would be used to select which of the 4 bytes in the MDR to start transferring into the destination register.
If the starting byte is not on the proper boundary, the MDR may not all of the required bytes. For example, a properly aligned word address would correspond to byte offset 0 within the MDR and all 4 bytes would be copied from MDR into the desitination register; but if the word address is not aligned then at least one of the desired bytes would not be in the MDR. This is why an exception is triggerred to indicate the inability to successfully complete the operation.

Module 10 Example Set 4

1. How does a DRAM differ from SDRAM?
SDRAM is short for synchronous DRAM. It runs in synchronization with the memory bus. Traditional DRAM used an asynchrono9us interface, which means it operated independently of the processor . Although the latency for the first unit of data is the same for both, the fact that the SDRAM signals are synchronized with the system bus signals allows SDRAM to supply additional data units within a burst transfer in fewer cycles. For example, acquiring the first or 4 data units within a burst may take 5 cycles, but each of the remaining data units within the burst would take only 1 cycle per unit for a total of 8 (5+1+1+1) cycles. Asynchronous DRAM on ther otherhand could take as much 14 cycles (5+3+3+3) to supply the same amount of data.

2. How does DDR, DDR2, DDR3 and DDR4 SDRAM differ from standard SDRAM.
Regular or standard SDRAM is often called single data rate (SDR) because it transfers a single unit of data per transfer cycle. DDR (double data rate) SDRAM provides two units of data per transfer cycle by sending data at the leading and trailing edge of the bus clock cycle. So DDR essentially operates at a frequency that is double that of the data bus frequency.
DDR2 (DDR generation 2) acheives twice the throughput of DDR memory by using differential pairs of signal wires to allow faster signaling without noise and interference problems. Its internal operating frequency is twice that of DDR and it can perform up to 1066MTps (millions of transfers per second).
Shown below is an image of a typical DDR2 memory module:



DDR3 has higher levels of performance, lower power consumption and greater reliability than DDR2. It cuts the internal clock cycle time again to half that of the DDR2 memory and employs a lower operating voltage. It provides a transfer rate of up to 2133MTps.
DDR4 takes this a step further by operating at an internal frequency twice that of DDR3 and operating at an even lower voltage level to reduce power consumption and save energy by producing less heat. It provides a transfer rate of up to 4266MTps.

3. How is the width of a memory chip and the width of the CPU-to-memory bus related to the amount of data that is obtained in response to a read operation?
The width of the cpu-to-memory bus must match the sum of the widths of the memory chips that are accessed in parallel. For example, if 4 memory chips are selected together and accessed in parallel and each chip has a width of 8, then the bus width would have to be 4*8 = 32 bits. This determines the amount of data that is transferred in response to a single memory read operation.

4. Describe the behavior of a MIPS lb (load byte) instruction?
In executing the lb instruction, the sum of the contents of the base register plus the sign-extended offset is used to identify the location in memory of the byte to be transferred. Bits 2 through 31 of the address identify the memory word to be read. The low 2 bits of the memory address (bits 0 and 1) are used as the offset to the byte within the word to be used. This byte is copied into the lower 8-bits of the result register and the upper 24 bits are filled with copies of the sign bit (the MSB) from the byte.

5. How does the behavior of a MIPS lbu (load byte unsigned) instruction differ from the behavior of a lb (load byte) instruction?
The only difference is that the upper 24 bits of the result register are filled with 0's by the lbu instruction rather than with copies of the sign bit in the loaded byte.

6. What determines the amount of data that is obtained from a memory chip in response to a single read operation?
The size of each memory cell (i.e., the width of the chip) determines how much data will be obtained from the chip in response to a single read operation.

7. What determines the amount of data the is transferred over the cpu-to-memory bus in response to a single read operation?
The width of the bus determines how much data will be transferred in response to a single read operation. The width of the bus is some multiple of the width of the memory chips. For example, if the width of each chip is 16 bits and the width of the bus is 32 bits, then two chips must be read in parallel to obtain 32 bits of data. If instead, the bus width is 16 bits, then only one chip of width 16 would be read.

8. Given the width and depth of a memory chip, how is the total storage capacity of the chip (in bytes) computed?
The storage capacity in bits is just the product of the width times the depth. The width is the number of bits per cell and the depth is the number of cells in the chip. The storage capacity in bytes is given by (width*depth)/8 since a byte contains 8 bits.