CS220: Computer Organization
Mid-semester Examination Solution Sketch

- Please be concise in your answers and write legibly.
- Do not write unnecessarily long answers. Longer answers usually lead to lower scores.
- Every answer must be accompanied by adequate explanation. Any answer (correct or incorrect) without explanation will not receive any credit.
- This is an open-book/open-note examination.

1(a). We would like to design an SRAM module of capacity $2^{22} \times 32$ bits. The environment interacts with the SRAM module by sending a 22-bit address and expects a 32-bit data in return. Internally, the SRAM module is designed using SRAM chips each having 2^{19} rows and 8-bit width. A read access to a chip outputs one row i.e., 8 bits. Consecutive bytes are placed on consecutive chips i.e., byte#0 on row#0 of chip#0, byte#1 on row#0 of chip#1, ..., byte#N-1 on row#0 of chip#N-1, byte#N on row#1 of chip#0, and so on where the total number of chips is N. Suppose the environment sends the hexadecimal address 22'hDEADA to the SRAM module. Show how this address is used to internally select an appropriate number of chips and access their rows to assemble the correct 32-bit output. Assume that each chip has a chip select input. Your answer must include a rough sketch of the hardware. **(5 points)**

Solution: Number of chips is 32. Since four consecutive chips store a 32-bit value, we can group the chips into eight groups. A group is selected based on the least significant three bits of the 22-bit address. The chip select inputs of all chips in a group are enabled together when that group is selected. The most significant 19 bits of the address are used to access a row in the chips of the selected group. Note that the group selection bits must be the least significant bits of the address because consecutive bytes are placed on consecutive chips. The hardware includes two decoders. One decoder will take three least significant bits of the address and generate the chip select inputs of all chips in a group i.e., one output of the decoder will be connected to all chip select inputs of a group and there are eight groups. The second decoder takes the remaining 19 bits of the address and generates the wordline signals for the SRAM chips.

1(b). To improve the access time to a chip in the above question, the designers decide to size each chip to have 2^{18} rows and 16-bit width. However, a read access to a chip continues to output only 8 bits (the left or right half of a row depending on the column address). Consecutive bytes continue to be placed on consecutive chips i.e., byte#0 on left half of row#0 of chip#0, byte#1 on left half of row#0 of chip#1, ..., byte#N-1 on left half of row#0 of chip#N-1, byte#N on right half of row#0 of chip#0, byte#N+1 on right half of row#0 of chip#1, ..., byte#2N-1 on right half of row#0 of chip#N-1, byte#2N on left half of row#1 of chip#0, and so on where the total number of chips is N. Suppose the environment sends the hexadecimal address 22'hDEADA to the SRAM module. Show how this address is used to internally assemble the correct 32-bit output. Your answer must include a rough sketch of the hardware. **(10 points)**

Solution: The least significant three bits of the address continue to be the group selection bits. Each group continues have four chips. The next bit of the address (A[3]) selects the left or right halves of the outputs from the four selected chips. If this bit is zero, left half is selected and if this bit is one, the right half is selected. The remaining 18 bits of the address are used to access a row of the chips in the selected group. The hardware includes three decoders. One decoder will take three least significant bits of the address and generate the chip select inputs of all chips in a group i.e., one output of the decoder will be connected to all chip select inputs of a group and there are eight groups. The second decoder takes the most significant 18 bits of the address and generates the wordline signals for the SRAM chips. The third decoder takes A[3] and generates the output enable for left and right halves.

2. Consider a 32-bit MIPS processor with an SRAM cache and a DRAM module. A subset of the DRAM contents relevant to this question are shown in Table 1. Each byte in the first 2^{22} addresses has value 8'hAA, each byte in the next 2^{22} addresses has value 8'hBB, each byte in the next 2^{22} addresses has value 8'hCC, each byte in the next 2^{22} addresses has value 8'hDD, each byte in the next 2^{22} addresses has value 8'hEE. Consider the MIPS

Table 1. Values in DRAM

Address	Value
0	8'hAA
1	8'hAA
...	8'hAA
$2^{22}-1$	8'hAA
2^{22}	8'hBB
$2^{22}+1$	8'hBB
...	8'hBB
$2\times 2^{22}-1$	8'hBB
2×2^{22}	8'hCC
$2\times 2^{22}+1$	8'hCC
...	8'hCC
$3\times 2^{22}-1$	8'hCC
3×2^{22}	8'hDD
$3\times 2^{22}+1$	8'hDD
...	8'hDD
$4\times 2^{22}-1$	8'hDD
4×2^{22}	8'hEE
$4\times 2^{22}+1$	8'hEE
...	8'hEE
$5\times 2^{22}-1$	8'hEE

instruction `lh $2, -16($29)`, where the displacement is written in decimal. Assume that the displacement is a 32-bit value. The value stored in $29 is 32'h1000000. Assume two's complement arithmetic throughout.

2.(a) What value will $2 contain after this instruction completes execution? Briefly explain. **(5 points)**

Solution: The half word will be loaded from byte addresses $2^{24}-16$ and $2^{24}-15$. Both these addresses contain 8'hDD. Therefore, the half word contains 16'hDDDD. Since the instruction is `lh`, the half word will be sign-extended before loading in $2. So, $2 will contain 32'hFFFFDDDD.

2.(b) Suppose that the address encoded in this instruction is not found in the SRAM cache and therefore, a read request must be sent to the DRAM for fetching the data. The data interface between the SRAM cache and the DRAM controller is 64-byte wide. Assume that the DRAM controller always sends an address that is aligned to a

64-byte boundary to the DRAM module (i.e., the address is divisible by 64). The DRAM module has two channels, four ranks per channel, eight x16 chips per rank, eight banks per chip, and each bank has 8192 rows and 1024 columns, where each column is 16-bit wide matching the output width of a chip. What should be the burst length to achieve the fastest completion of a request sent from the DRAM controller? **(2 points)** What will happen if the burst length is more or less than this? Briefly explain. **(2+2 points)** Write down the address in hexadecimal that the DRAM controller will send to the DRAM to complete the `lh` instruction. **(2 points)** What are the row number, rank number, bank number, column number, and channel number corresponding to this address? **(2+2+2+2+2 points)**

Solution: Since each rank has eight x16 chips, the channel width is 128 bits. Therefore, gathering 64 bytes requires four bursts on the channel. Hence, the burst length is four. If the burst length is more, more than 64 bytes of data will be transferred to the SRAM cache. If the additional data is not required by the processor, this will unnecessarily hold up the channel for a longer period of time delaying the subsequent requests to the DRAM (note that this will not delay the current request because the DRAM controller keeps transferring the bursts to the processor as and when they come out from DRAM). If the burst length is less, multiple requests have to be sent by the DRAM controller to the DRAM to complete a 64-byte request increasing the latency of every request. Since $2^{24} - 16$ is 0xFFFFF0 and the address must be aligned to 64-byte boundary, the address sent to the DRAM is 0xFFFFC0. Since the channel width is 16 bytes, the least significant four bits of the address refer to one burst. The next two bits are column-low, the next bit is channel, the next eight bits are column-high, the next three bits are bank number, the next two bits are rank number, and the most significant thirteen bits are row number. The address 0xFFFFC0 corresponds to row=15, rank=3, bank=7, column=1020, and channel=1.

3.(a) Translate the following code snippet written in C programming language into 32-bit MIPS assembly language. Assume that `char` is one byte and `short` is two bytes. The starting addresses of `x`, `y`, `a`, `b` are in registers \$1, \$2, \$3, \$4, respectively. Further, the variables `z1`, `z2`, `c1`, `c2` are allocated in \$5, \$6, \$7, and \$8, respectively. **(16 points)**

```
char x[100], y[100], z1, z2;
unsigned short a[10], b[10], c1, c2;

z1 = x[21];
z2 = x[50];
z1 = z1 + z2;
y[79] = z1;
c1 = a[0];
c2 = b[3];
c1 = c2 - c1;
a[9] = c1;
```

Solution: The translation is given below.

```
lb  $5, 21($1)
lb  $6, 50($1)
add $5, $5, $6
sb  $5, 79($2)
lhu $7, 0($3)
lhu $8, 6($4)
sub $7, $8, $7
sh  $7, 18($3)
```

3.(b) In the aforementioned code snippet, if all elements of array `x` are initialized to -20, all elements of array `a` are initialized to -100, and all elements of array `b` are initialized to 100, what are the contents of registers \$5, \$6, \$7, and \$8 in hexadecimal at the end of the code snippet? **(8 points)** What are the contents of `y[79]` and `a[9]` in hexadecimal at the end of the code snippet? **(4 points)** Assume two's complement arithmetic. All your answers must be in correct two's complement representation.

Solution: After the `lb` instructions, both \$5 and \$6 contain sign-extended -20 i.e., 0xffffffec. This is the final value in \$6. After the `add` instruction, the value in \$5 changes to 0xffffffd8. This is the final value in \$5. The `sb` instruction stores only the least significant byte of \$5. So, `y[79]` will have 0xd8. After the `lhu` instructions, \$7 contains zero-extended -100 i.e., 0xff9c and \$8 contains zero-extended 100 i.e., 0x64. This is the final value in \$8. After the `sub` instruction, the value in \$7 changes to 0xffff00c8. This is the final value in \$7. The `sh` instruction stores only the least significant two bytes of \$7. So, `a[9]` will have 0xc8.

4. Suppose we would like to represent the decimal real number -4.3 in IEEE 754 single-precision format. Derive the representation for each of the following four rounding modes: round toward $+\infty$, round toward $-\infty$, round toward zero, and round to the nearest with the IEEE 754 rule for halfway rounding. **(16 points)**

Solution: $-4.3 = -100.0100110011001\ldots = -1.00010011001\ldots \times 2^2$. The biased exponent is 127+2 or 129. Rounding toward $+\infty$ and zero will round the mantissa to the smaller magnitude of the two neighboring values i.e., $-1.00010011001100110011001 \times 2^2$ because the number is negative. Rounding toward $-\infty$ will round the mantissa to the larger magnitude of the two neighboring values i.e., $-1.00010011001100110011010 \times 2^2$ because the number is negative. Rounding to the nearest will also lead to the same representation as rounding toward $-\infty$ because this representation has the 23rd bit of the mantissa equal to zero.

5. Table 2 shows the next state function of a finite state machine having thirteen states. The machine takes a two-bit input (Y[1:0]). The output function is not specified and can be ignored for this question.

Table 2. Next state function

Current state	Input (Y[1:0])	Next state
S_0	2'bxx	S_1
S_1	2'bxx	S_2
S_2	2'bxx	S_3
S_3	2'b00	S_4
S_3	2'b01	S_5
S_3	2'b1x	S_6
S_4	2'bxx	S_7
S_5	2'bxx	S_7
S_6	2'bxx	S_7
S_7	2'bxx	S_8
S_8	2'bxx	S_9
S_9	2'bxx	S_{10}
S_{10}	2'b00	S_{11}
S_{10}	2'b01, 2'b1x	S_{12}
S_{11}	2'bxx	S_0
S_{12}	2'bxx	S_0

5.(a) Draw the state transition diagram. **(3 points)**

Solution: This is easily constructed from the table.

5.(b) If the next state function is entirely stored in a microcode ROM without taking help from any auxiliary dispatch ROMs and state incrementer, calculate the number of rows and columns of the microcode ROM. Briefly explain (there is no need to show the full truth table). **(5 points)**

Solution: The input to the microcode ROM would be the current state and Y[1:0]. Since there are thirteen states and for each state, there can be four possible Y values, the ROM would have 52 different inputs leading to 52 rows. Each row must store the corresponding next state, which requires four bits. So, the ROM must have four columns.

5.(c) Suppose a state incrementer can be used to compute the next state along with two dispatch ROMs, a microcode ROM, and a state selection multiplexer. The microcode ROM takes only the current state as input to look up a row and outputs the branch control for computing the next state (note that Y cannot be used as an input to the microcode ROM). The two dispatch ROMs are used to encode the two branches in the state diagram. The first branch occurs when the current state is S_3 and the second branch occurs when the current state is S_{10}. The first dispatch ROM stores the next states when the current state is S_3. The second dispatch ROM stores the next states when the current state is S_{10}. The input Y is used to look up a row in the dispatch ROMs. Show the contents of the microcode ROM and the two dispatch ROMs row by row. **(6+3+3 points)** Show the inputs to the state selection multiplexer. **(8 points)**

Solution: Let us denote the branch controls for incremented state as 0, for S_3 dispatch ROM as 1, for S_{10} dispatch ROM as 2, for converging back from S_4 and S_5 to S_7 as 3, and for converging back from S_{11} and S_{12} to S_0 as 4. The contents of the ROMs are shown below. The microcode ROM is indexed by the current state and outputs the branch control. The dispatch ROMs are indexed by Y[1:0] and output the next state. The multiplexer's selection

Table 3. Microcode ROM

0
0
0
1
3
3
0 or 3
0
0
0
2
4
4

Table 4. Dispatch ROM for next state of S_3

0100
0101
0110
0110

lines are tied to the output lines of the microcode ROM. Since the microcode ROM generates three-bit output, there are three selection lines. However, the selection lines will attain only five possible values corresponding to

Table 5. Dispatch ROM for next state of S_{10}

1011
1100
1100
1100

the branch control output of the microcode ROM: 0, 1, 2, 3, 4. The corresponding five inputs to the multiplexer are respectively current state+1, S_3 dispatch ROM output, S_{10} dispatch ROM output, 0111, and 0000. Since the multiplexer selects the next state, each of the five inputs to the multiplexer is four-bit wide.