Assignment 2

Computer Architecture CS3350B

Albirawi, Zaid
250626065

March 2nd, 2015

1. Consider a 2-way-set-associative cache with a 32-bit address, including 4-bit data block offset and 6-bit index. Assume that a word consists of fours bytes. Starting for power on, the following word-addressed cache references are recorded.

   11, 26, 21, 52, 23, 30, 47, 6, 35, 44, 54, 5

   Use LRU (least recently used) replacement.

   1.1. List the final state of the cache, with each valid entry represented as a record of <index, tag, data block(s)>.

Reference	**Tag** [0 - 21]	**Index** [22 - 27]	**Offset** [28 - 31]
11	00000000000000000000	000000	1011
26	00000000000000000000	000001	1010
21	00000000000000000000	000001	0101
52	00000000000000000000	000011	0100
23	00000000000000000000	000001	0111
30	00000000000000000000	000001	1110
47	00000000000000000000	000010	1111
6	00000000000000000000	000000	0110
35	00000000000000000000	000010	0011
44	00000000000000000000	000010	1100
54	00000000000000000000	000011	0110
5	00000000000000000000	000000	0101

Index	Tag	Data block (8 words)			
		3	2	1	0
000000	0	Mem[11]	Mem[10]	Mem[09]	Mem[08]
000000	0	Mem[07]	Mem[06]	**Mem[05]**	Mem[04]
000001	0	Mem[31]	Mem[30]	Mem[29]	Mem[28]
000001	0	**Mem[23]**	Mem[22]	Mem[21]	Mem[20]
000010	0	Mem[47]	Mem[46]	Mem[45]	**Mem[44]**
000010	0	Mem[35]	Mem[34]	Mem[33]	Mem[32]
000011	0	Mem[55]	**Mem[54]**	Mem[53]	Mem[52]
000011	-	-	-	-	-

##, <index, tag, data block>
11, <000000, 00000000000000000000, 3>
26, <000001, 00000000000000000000, 2>
21, <000001, 00000000000000000000, 1>
52, <000011, 00000000000000000000, 0>
23, <000001, 00000000000000000000, 3>
30, <000001, 00000000000000000000, 2>
47, <000010, 00000000000000000000, 3>
06, <000000, 00000000000000000000, 2>
35, <000010, 00000000000000000000, 3>
44, <000010, 00000000000000000000, 0>
54, <000011, 00000000000000000000, 2>
05, <000000, 00000000000000000000, 1>

1.2. What is the hit ratio?

The hit ratio is $4 \div 12 = 33.33\%$

2. We would like to expand the MIPS register file to 128 registers (rather than usual 32) and expand the instruction set by letting the ‘op’ field be 20 bits. Assume that a word consists of eight bytes.

2.1. Assume that the ‘op’ field determines the function field (assume that the number of bits for ‘funct’ and ‘op’ are the same). How would this affect the size of each of the bit fields in the R-type instructions? Label all the fields with their name and bit length and explain the consequences.

Assuming that the instruction word is 8 bytes, 64 bits.

opcode	rs	rt	rd	shamt	funct
20 bits	7 bits	7 bits	7 bits	3 bits	20 bits

All register fields must have 7 bits for them to be able to address all 128 registers as $2^7 = 128$. Furthermore, the shamt field will only receive $64 - 20 - 20 - 7 - 7 - 7 = 3$ bits, as those are the only bits leftover from the 64 bit instruction word. This will limit the user to a maximum of $2^3 = 8$ bits shift for every instruction.

2.2. How would this affect the size of each of the bit fields in the I-type instructions? Label all the fields with their name and bit length and explain the consequences.

Assuming that the instruction word is 8 bytes, 64 bits.

opcode	rs	rt	constant or address
20 bits	7 bits	7 bits	30 bits

All register fields must have 7 bits for them to be able to address all 128 registers as $2^7 = 128$. Furthermore, the address field will receive $64 - 20 - 7 - 7 = 30$ bits, as those are the bits leftover from the 64 bit instruction word. Therefore, the instruction will be able to support -2^{29} to $2^{29} - 1$ different constant address values or a 30 bit address offset value.

3. Consider the following MIPS loop:

```
LOOP: slt  $t2, $0, $t1
      beq  $t2, $0, DONE
      srl  $t1, $t1, 1
      addi $s1, $s1, 1
      j    LOOP
DONE:
```

3.1. Assume that register $t1 is initialized to the value 10. What is the value in register $s1 assuming $s1 is initially zero?

The value of $s1 will be 4, as the program will run until $t1 is equal to 0. Furthermore, since the integer 10 is equal to 1010 in binary then the program requires four srl instructions to shift the value of $t1 to zero.

3.2. For the above loop, write the equivalent C code routine. Assume that the registers $s1, $t1, and $t2 are integers A, i, j, respectively.

```
for ( ; 0 < i; i /= 2)
     A++;
```

3.3. Assume that the register $t1 initialized to the value N. How many MIPS instructions are executed?

Every loop the program successfully completes will execute 5 instructions, assuming none of the instructions above are pseudo instructions. In addition, the last loop of the program will only execute 2 instructions, the slt and beq instructions. Furthermore, the program will run until it the highest N bit set is shifted to the right. Therefore, the amount of MIPS instructions that will execute when the value is set to N is $\left[\left(\lfloor log_2{}^N \rfloor + 1\right) * 5\right] + 2$.

4. Consider the following C code function to compute the Fibonacci number:

```
int fib (int n) {
     if (n < 2)
          return 1;
     else
          return fib (n - 1) + fib (n - 2)
}
```

Then starting from n = 0, the sequence will look like

1, 1, 2, 3, 5, 8, 12, 21, 34, 55, 89, 144, …

4.1. Implement the above C code in MIPS assembly. Try to use a minimal number of MIPS instructions.

```
fib:
lw    $ra, 8($sp)      #loads the return address
addi  $sp, $sp, 12     #frees the stack space
jr         $ra         #returns to the line it was called from

fib_rec:
addi  $sp, -12         #allocates space on the stack for $ra, $a0, $v0
sw    $ra, 8($sp)      #stores the return address number on the stack

slti  $v0, $a0, 2      #sets the value of v0 to 1 if n is less than 2
beq   $v0, 1, fib      #branches to fib_rec if v0 is equal to 1

sw    $a0, 0($sp)      #stores the value of n
addi  $a0, -1          #n-1, first child
jal   fib_rec          #recursive call
sw    $v0, 4($sp)      #stores the result of the first child

lw    $a0, 0($sp)      #loads the value of n at that tree level
addi  $a0, -2          #n-2, second child
jal   fib_rec          #recursive call

lw    $v1, 4($sp)      #loads the result of the first child into $v1
add   $v0, $v0, $v1    #calculates the result
j     fib              #returns the main function
```

4.2. Write a complete MIPS code and run it in QtSpim to compute fib(20). Show the final values of the registers used.

```
main:
addi  $sp, -4           #allocates space on the stack for the $ra
sw    $ra, 0($sp)       #stores the return address number on the stack

addi  $a0, $0, 20, -1 #set the n value
jal   fib_rec           #jumps to the fib_rec label

lw    $ra, 0($sp)       #returns the return address number to $ra
addi  $sp, -4           #frees the space that was in use on the stack
jr    $ra               #return call
```

$v0 = 6765**
$v1 = 4181
$a0 = 1

**QtSpim changes the value of $v0 to 10 after the program finishes executing.

5. Assume that we have a machine which is byte-addressable machine, and a word consists of four bytes. The table below shows 32-bits values of an array stored in the memory.

5.1. Write C code to find maximum value among the above data. Assume that the data shown represents the C variable called A. Which is an array typeof int.

```
max = A[0];
for (int i = 1; i < sizeof(A)/sizeof(A[0]); i++)
      max = (max > A[i] ? A[i] : max);
```

5.2. Write MIPS code to find the maximum value among the above data. Use minimum number of MIPS instructions as possible. Assume the base address of A is stored in register #s4.

```
find_max:
lw          $v0, 0($s0)       #load A[6] into $a0

LOOP:
addi        $s0, 4            #shift the address to the next integer
lw          $t0, 0($s0)       #loads the next value

beq         $t0, -1, TERM     #terminates
slt         $t1, $v0, $t0     #sets $t1 if a new max was found
bne         $t1, $0,new_max   #jumps to new_max label

j           LOOP              #loops

TERM:
jr          $ra               #terminates

new_max:
addi        $v0, $t0, 0       #sets the new value to $v0
j           LOOP              #loops
```

5.3. Write a complete MIPS code and run it in QtSpim. Show final values of registers used.

```
main:
addi  $sp, $sp, -8            #allocate space for the $ra and $s0 values
sw          $ra, 4($sp)       #store $ra
sw          $s0, 0($sp)       #store $s0

la          $s0, ary          #load the address of the array into $s0
addi        $s0, 24           #shift to A[6], 24th bit
jal         find_max          #jump the find_max label

lw          $s0, 0($sp)       #load the $s0 original value
lw          $ra, 4($sp)       #load the $ra original value
addi        $sp, $sp, 8       #free the space allocated on the stack
jr          $ra               #return
```

...the code in 5.2…

```
            .data
ary:        .word 0, 0, 0, 0, 0, 0, 2, 4, 7, 3, 8, 6, 5, 1
end_ary:    .word -1
```

$v0 = 8**
$t0 = -1
$t1 = 0

**QtSpim changes the value of $v0 to 10 after the program finishes executing.

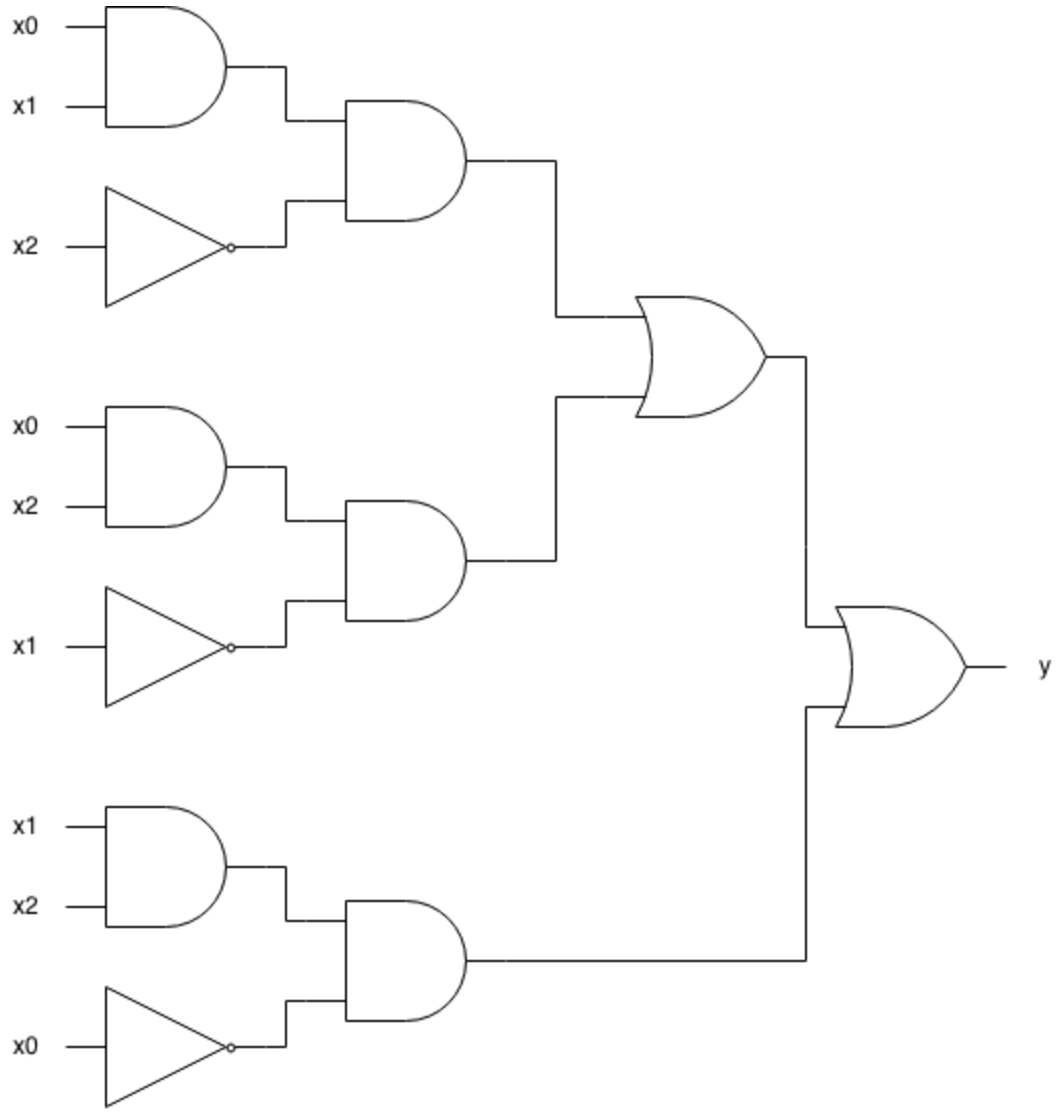
6. Assume that X consists of 3 bits: x2, x1, x0, and that y is represented by 1 bit. We expect that y is true if and only if X contains only two 1s.
   6.1. Show the true table with each entry represented by <x2 x1 x0 y>

x0	x1	x2	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

   6.2. Write a logical function y.

$$y = [(x0 \wedge x1) \wedge \neg x2] \vee [(x0 \wedge x2) \wedge \neg x1] \vee [(x1 \wedge x2) \wedge \neg x0]$$
$$y = [x0 \wedge x1 \wedge \neg x2] \vee [x0 \wedge x2 \wedge \neg x1] \vee [x1 \wedge x2 \wedge \neg x0]$$
$$y = \{x0 \wedge [(x1 \wedge \neg x2) \vee (x2 \wedge \neg x1)]\} \vee [x1 \wedge x2 \wedge \neg x0]$$
$$y = [x0 \wedge (x1 \oplus x2)] \vee [\neg x0 \wedge x1 \wedge x2]$$

6.3. Using Logisim, draw the gate diagram using AND, OR, and NOT gates only.



6.4. Using Logisim, draw the gate diagram using two AND gates, one XOR, NOT and OR gate, respectively.