CS2100 (AY2021/22 Semester 1) Answer Sheets

Student No: ________________________

(If you are using this file, remember to create a pdf file and rename it with your Student Number (eg: A1234567X.pdf). Write your answers in the box/space provided.)

1	D	2	B	3	A	4	C	5	A	6	D
7	A,B,D or A,B,C,D	8	C,D	9	A,D	10	B,D,E	11	B,E

Q12. Sequential circuits [12 marks]

(a) (i) [4]

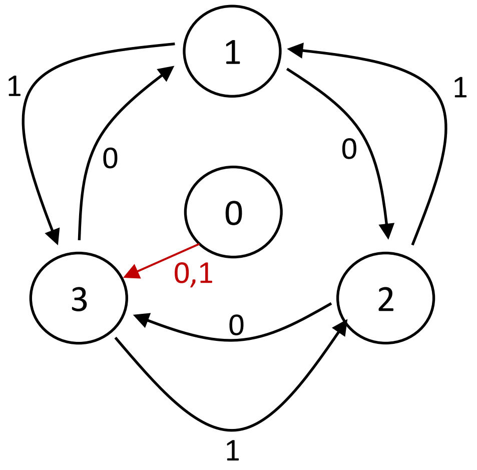
JA = 1

KA = $B' \cdot x + B \cdot x'$

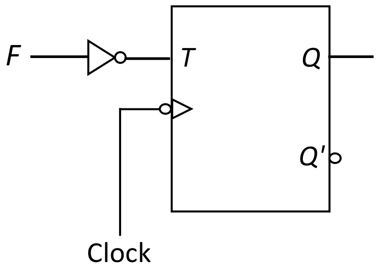
JB = 1

KB = $A' \cdot x' + A \cdot x$

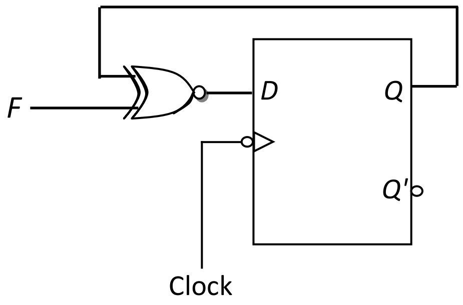
(ii) [2]



(b) (i) [3]



(ii) [3]



Q13. Combinational circuits [13 marks]

(a) [4]

$E(A,B,C,D) = \Sigma m$ (8, 9, 10, 11, 12, 13, 14, 15) or (8 – 15)

$F(A,B,C,D) = \Sigma m$ (4, 5, 6, 7, 8, 9, 10, 11) or (4 – 11)

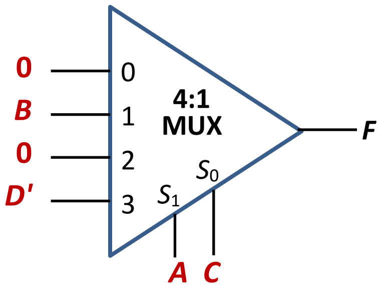
$G(A,B,C,D) = \Sigma m$ (2, 3, 4, 5, 10, 11, 12, 13)

$H(A,B,C,D) = \Sigma m$ (1, 2, 5, 6, 9, 10, 13, 14)

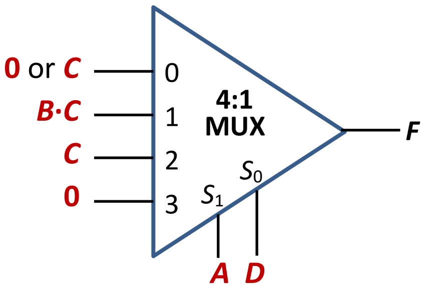
(b) [4]

$K = A' \cdot B' \cdot C$

(c) [5]



or



Q14. MIPS [12 marks]

(a) [2]

17

(b) [4]

```
for (answer = 0, k = 0; k < size; k++) {
   if (A[k] >= B[k])
      answer += A[k];
    else
      answer -= B[k];
}
```

(c) [2]

0x 1160000B

(d) [2]

0x 8D310000

(e) [3]

0x 0AC00404

Q15. Pipelining [14 marks]

(a) [2]	**21**	(b) [3]	**+15**	(c) [3]	**+8**	(d) [3]	**+4**

(e)
[3]

Answer:

One answer (other answers possible):

Move I5 (sll $t8, $a0, 2) to above I3 to reduce 2 cycles of delay.

1 mark: Which instruction to move.
1 mark: Where to move to.
1 mark: How many delay cycles are reduced.

Q16. Cache [18 marks]

(a)
[2]

Index: **6** ; Byte offset: **5**

(b)
[2]

Hit rate for array *A* = **7/8** ; Hit rate for array *B* = **7/8**

(c)
[4]

Hit rate for array *A* = **384/1028** or **96/257** ; Hit rate for array *B* = **384/1028** or **96/257**

(d)
[2]

Lowest hit rate for array *A* = **0**

How many elements in array *A* would result in this hit rate?

When array *A* has a multiple of 512 elements. (Other answers possible.)

(e)
[2+3]

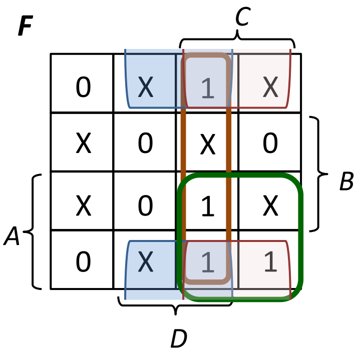
Set index: **1** ; Byte offset: **4**

Number of misses: **6**

(f)
[3]

Number of misses: **10**

Workings in the following pages.



0	X	1	X
X	0	X	0
X	0	1	X
0	X	1	1

Workings/Explanations

1. $(1022)_3 = 1\times3^3 + 2\times3 + 2 = (35)_{10} = 5\times7 = (50)_7$.

5. 4 PIs: $C{\cdot}D$, $B'{\cdot}D$, $B'{\cdot}C$ and $A{\cdot}C$.
   There are no EPIs.

7. In big endian the data represented is 0x43617473 (or 1130460275), in little endian this is 0x73746143 (or 1937006915).

   Strings are stored and read from lowest to highest memory addresses independent of endianness. Since 0x43 = ‘C’, 0x61 = ‘a’, 0x74 = ‘t’, 0x73 = ‘s’, this reads as “Cats”, regardless of whether the computer is little- or big-endian.

8.

addi \$3, \$zero, 0 # instruction 1
opcode = 001000; rs = 00000; rt = 00011; immed = 0000 0000 0000 0000
Instruction = 001000 00000 00011 0000 0000 0000 0000
= 0010 0000 0000 0011 0000 0000 0000 0000
= 0x20030000

addi \$3, \$3, 1 # instruction 3
opcode = 001000; rs = 00011; rt = 00011; immed = 0000 0000 0000 0001
Instruction = 001000 00011 00011 0000 0000 0000 0001
= 0010 0000 0110 0011 0000 0000 0000 0001
= 0x20630001

andi \$4, \$3, 1 # instruction 4
opcode = 001100 ; rs = 00011; rt = 00100; immed = 0000 0000 0000 0001
Instruction = 001100 00011 00100 0000 0000 0000 0001
= 0011 0000 0110 0100 0000 0000 0000 0001
= 0x30640001

bne \$4, \$zero, loop # instruction 5
opcode = 000101; rs = 00100; rt = 00000; immed = -3 = -(0000 0000 0000 0011) = (1111 1111 1111 1101)
Instruction = 000101 00100 00000 1111 1111 1111 1101
= 0001 0100 1000 0000 1111 1111 1111 1101
= 0x1480FFFD

j loop # instruction 7
The “loop” label is at address 8, or 0000 0000 0000 0000 0000 0000 0000 1000
Remove 4 MSB and 2 LSB: ~~0000~~ 0000 0000 0000 0000 0000 0000 10~~00~~
opcode = 000010
Instruction = 000010 0000 0000 0000 0000 0000 0000 10
= 0000 1000 0000 0000 0000 0000 0000 0010
= 0x08000002

10.

A. No, bitwise operations do a zero-extend instead of a sign-extend.

B. Yes. The 4 MSB from the program counter is taken from PC + 4 and not from PC. Hence for example we can have a j at location 00 00 00 00 00 00 00 00 to another j instruction at address 00 00 FF FF FF FF FF FC, which can then do a j to a target another 2^{26} instructions away at 00 01 FF FF FF FF FF FC, giving a total jump of up to 2^{27} instructions away.

C. No, branch displacements are always taken relative to the instruction after the branch regardless of direction.

D. Yes, the opcode is 6-bit long, so we can use a 6-to-64 decoder to decode it.

E. Yes, when the opcode is 000000, we use the 6-bit function code to define 2^6 = 64 instructions. Add to the remaining 63 opcodes from 000001 to 111111, gives us 63 + 64 = 127 instructions in total.

12.

(a)

Present state		Input	Next state		Flip-flop inputs			
A	*B*	*x*	A^+	B^+	*JA*	*KA*	*JB*	*KB*
0	0	0	X(1)	X(1)	X(1)	X(0)	X(1)	X(1)
0	0	1	X(1)	X(1)	X(1)	X(1)	X(1)	X(0)
0	1	0	1	0	1	X	X	1
0	1	1	1	1	1	X	X	0
1	0	0	1	1	X	0	1	X
1	0	1	0	1	X	1	1	X
1	1	0	0	1	X	1	X	0
1	1	1	1	0	X	0	X	1

13.

(b)

A	*B*	*C*	S_2	S_1	*K*
0	0	0	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	1	0

(c)

A	*B*	*C*	*D*	*F*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	X
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	X
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

14.

(c)

```
beq  $t3, $0, end
```

opcode = 0x4 = 0b000100

rs = $t3 = $11 = 0b01011; rt = $0 = 0b00000; end = 11

0b000100 01011 00000 0000000000001011 = **0x1160 000B**

(d)

```
lw  $s1, 0($t1)
```

opcode = 0x23 = 0b100011

rs = $t1 = $9 = 0b01001; rt = $s1 = $17 = 0b10001.

0b100011 01001 10001 0000000000000000 = **0x8D31 0000**

(e)

```
j loop
```

opcode = 0x2 = 0b000010

Address at loop: 0x2B00 0FFC + 0x14 = 0x2B00 1010

= 0b ~~0011~~ 1011 0000 0000 0001 0000 0001 00~~00~~

0b000010 1011 0000 0000 0001 0000 0001 00= **0x0AC0 0404**

15.

```
                                              (b)    (c)    (d)
       add  $t0, $0, $0     # I1
       add  $t9, $0, $0     # I2
       addi $t1, $a1, 0     # I3
       addi $t2, $a2, 0     # I4
       sll  $t8, $a0, 2     # I5
loop:  slt  $t3, $t0, $t8   # I6      +2
       beq  $t3, $0, end    # I7      +2            +1
       lw   $s1, 0($t1)     # I8      +3     +3
       lw   $s2, 0($t2)     # I9
       slt  $t3, $s1, $s2   # I10     +2     +1     +1
       bne  $t3, $0, else   # I11     +2            +1
       add  $t9, $t9, $s1   # I12     +3     +3
       j    cont            # I13
~~else: sub  $t9, $t9, $s2   # I14~~
cont:  addi $t0, $t0, 4     # I15     +1     +1     +1
       addi $t1, $t1, 4     # I16
       addi $t2, $t2, 4     # I17
       j    loop            # I18
end:
                       Total:         +15    +8     +4
```

16.

(a) (2 marks) Index: **6 bits;** byte offset: **5 bits**.

512/8 = 64 blocks → **6 bits** for index field; 32 bytes (8 words) per block → **5 bits** for byte offset.

(b) (2 marks) **7/8** for both arrays *A* and *B*. (1 mark each for *A* and *B*)

Starting address of *A* = 0xCDEF 0400 = 0b…. 0000 0100 0000 0000. Therefore *A*[0] resides in word 0 of block 32.

Array *A* contains 1032 elements. Since the cache consists of 512 words, *A*[1024] to *A*[1031] will occupy block 32. Therefore, *B*[0] will occupy word 0 of block 33.

Since there is no cache thrashing, for every 8 elements, the first would be a miss and the rest hits. Therefore, the hit rate is 7/8 for both arrays.

(c) (4 marks) **384/1028** or **96/257** for both arrays *A* and *B*. (2 marks each for *A* and *B*).

Array *A* contains 1028 elements. So, *A*[1024] to *A*[1027] will occupy the first 4 words of block 32, and *B*[0] to *B*[3] will occupy the next 4 words in the same block.

Therefore, for every 8 elements, the first 5 will be misses and the next 3 are hits, and this applies to the first 1024 elements. For the last 4 elements, they are all misses due to thrashing. Hence, there are 3/8 * 1024 = 384 hits altogether for each array.

A snapshot of the first 16 elements:

Block 32	A0 (M)	A1 (M)	A2 (M)	A3 (M)	A4 (M) B0 (M)	A5 (H) B1 (M)	A6 (H) B2 (M)	A7 (H) B3 (M)
Block 33	B4 (M) A8 (M)	B5 (H) A9 (M)	B6 (H) A10 (M)	B7 (H) A11 (M)	B8 (M) A12 (M)	B9 (M) A13 (H)	B10 (M) A14 (H)	B11 (M) A15 (H)
Block 34	B12 (M)	B13 (H)	B14 (H)	B15 (H)				

(d) (2 marks) (i) **0**. This occurs with cache trashing, i.e. the first elements of array *A* and array *B* are mapped to the same cache location. (ii) When array *A* has a multiple of 512 elements (this is the simplest answer).

(e) (2 marks) (i) Set index: **1 bit**; byte offset: **4 bits**.

16/4/2 = 2 sets → **1 bit** for set index field; 16 bytes (4 words) per block → **4 bits** for byte offset.

(3 marks) (ii) **6 misses**.

Address of I1 = 0x0x2B00 0FFC = 0b…. 1111 1100. So I1 is stored in set 1, word 3.

Execution: 1st iteration: I1 to I13, I15 to 18; 2nd – 100th iterations: I6 to I13, I15 to I18; I6 and I7.

Set 0	I2 (M) I18 (M)	I3 (H)	I4 (H)	I5 (H)	I10 (M)	I11 (H)	I12 (H)	I13 (H)
Set 1		I15 (M)	I16 (H)	I1 (M) I17 (H)	I6 (M)	I7 (H)	I8 (H)	I9 (H)

6 misses in the first iteration, thereafter all are hits.

(f) (3 marks) **10 misses**

Set index: 2 bits; byte offset: 3 bits.

Address of I1 = 0x0x2B00 0FFC = 0b…. 1111 1100. So I1 is stored in set 3, word 1.

Execution: 1st iteration: I1 to I13, I15 to 18; 2nd – 100th iterations: I6 to I13, I15 to I18; I6 and I7.

Set 0	I2 (M) I18 (M)	I3 (H)	I10 (M)	I11 (H)
Set 1	I4 (M)	I5 (H)	I12 (M)	I13 (H)
Set 2	I6 (M)	I7 (H)		I15 (M)
Set 3	I16 (M)	I1 (M) I17 (H)	I8 (M)	I9 (H)

10 misses in the first iteration, thereafter all are hits.