NATIONAL UNIVERSITY OF SINGAPORE

CS2100 – COMPUTER ORGANISATION

(Semester 2: AY2019/20)

ANSWER BOOKLET

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This answer booklet consists of **EIGHT (8)** printed pages.
2. Fill in your Student Number below. Do NOT write your name.
3. Make sure your answers are clearly written/typed.

STUDENT NUMBER (fill in with a pen):

A	**0**							

For examiner's use only		
Question	***Total***	***Marks***
Q1	16	
Q2	4	
Q3	10	
Q4	12	
Q5	12	
Q6	12	
Q7	14	
Q8	20	
Total	**100**	

Write your answers in the box/space provided.

1a. [2]

$s1 = **4A**

1b. [4]

Max $s1 = **78**

Initial $s0 = **0xFFFFFFFF**

1c. [6]

```
$s1 = 0;
while ($s0 != 0) {
  $t0 = $s0 % 16;
  $s1 = $s1 + $t0;
  $s0 = $s0 / 16;
}
```

Also accept: $s0 > 0
Also accept: $s0 = $s0 & 16;
Can combine with above: $s1 += $s0 %16;
Also accept $s0 = $s0 >> 4; (though not entirely correct due to arith vs logical shift.

1d. [2]

```
srl $s0, $s0, 4
```

0x00108102

1e. [2]

```
bne $s0, $zero, L
```

0x1600FFFC

Q1: _____ /16

---

2. [4]

-1.875 = **0xBFF00000**

Q2: _____ /4

3a. [3]

```
srl  $t0, $s0, 26
slti $t0, $t0, 1
```

Also accept 3 lines solution.

3b. [3]

```
srl $t1, $t1, 21
```

Also accept 2 lines solution such as:

```
sll $t1, $s1, 6
srl $t1, $t1, 27
```

3c. [4]

```
slti $t0, $s2, 1 # R-format
slti $t1, $s2, 5 # R & beq
add  $t2, $t0, $t1
```

← shortest solution without branch

usual solution with branch
|
v

```
        addi $t2, $zero, 2
        beq  $s2, $zero, end
        addi $t2, $zero, 1
        andi $t3, $s2  , 4
        beq  $t3, $zero, end
        addi $t2, $zero, 0
end:
```

Q3: _____ /10

4a. [4]

JA = **1**

KA = **1**

JB = ***A***

KB = ***A***

4b.(i) [4]

Number of states: **8**

0000 → 1000 → **1100 → 1110 → 1111 → 0111 → 0011 → 0001**

4b.(ii) [4]

State(1) = ***A·B'***

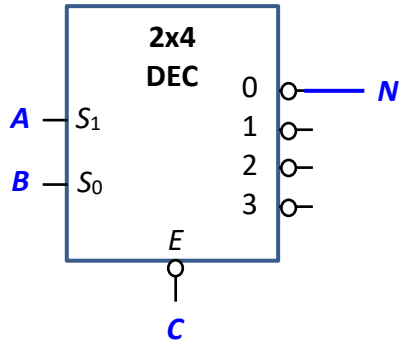
State(2) = ***B·C'***

Q4: /12

5a. (i) [2]

N = ***A + B + C***

(ii) [4]



A, *B*, *C* can be interchanged.

5b. [6]

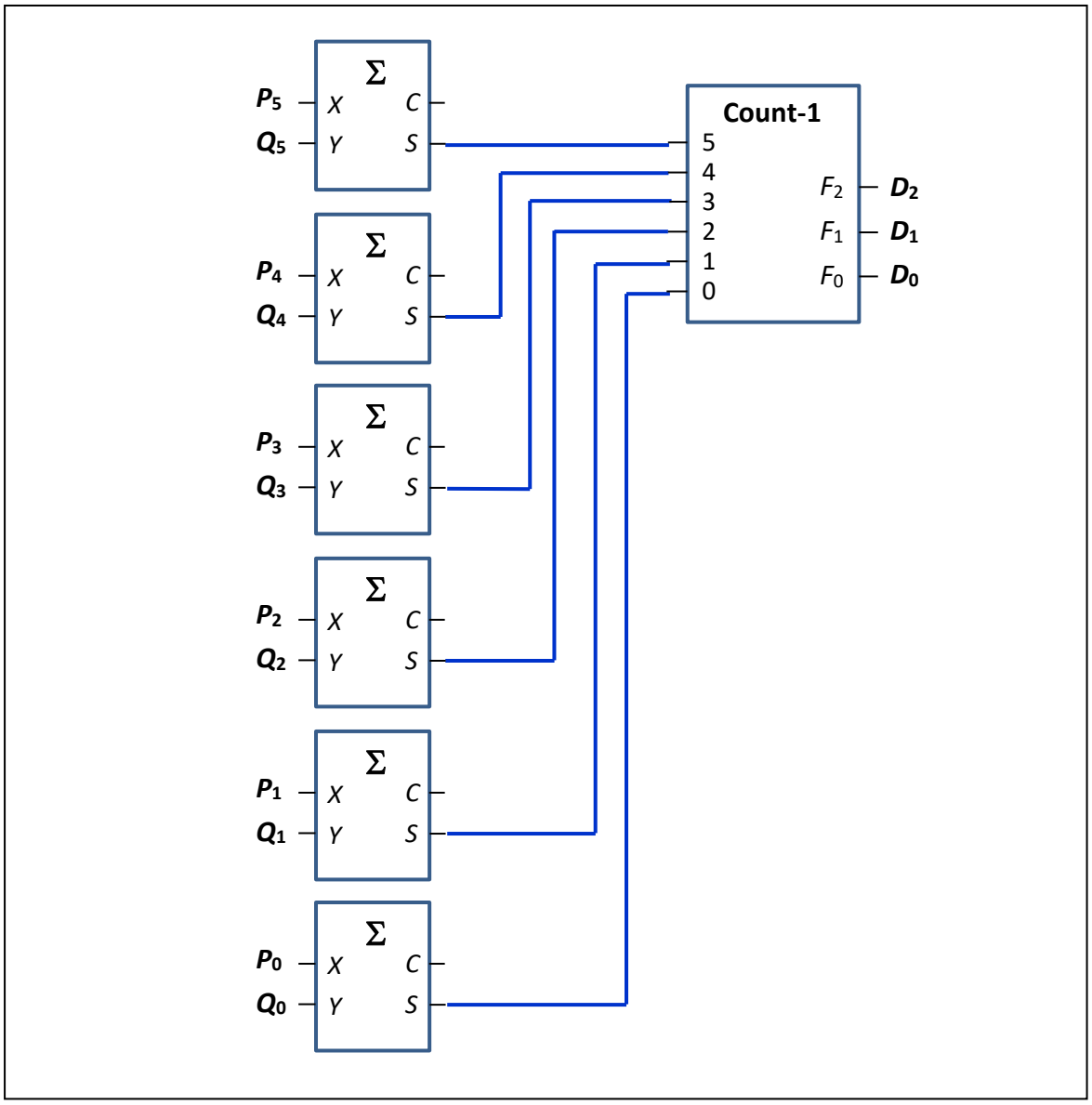
P = ~~*A·B·D + A·C·D'*~~ ***A·B·C·D + A·B'·C·D'***

Q = ***A'·C + B'·C·D***

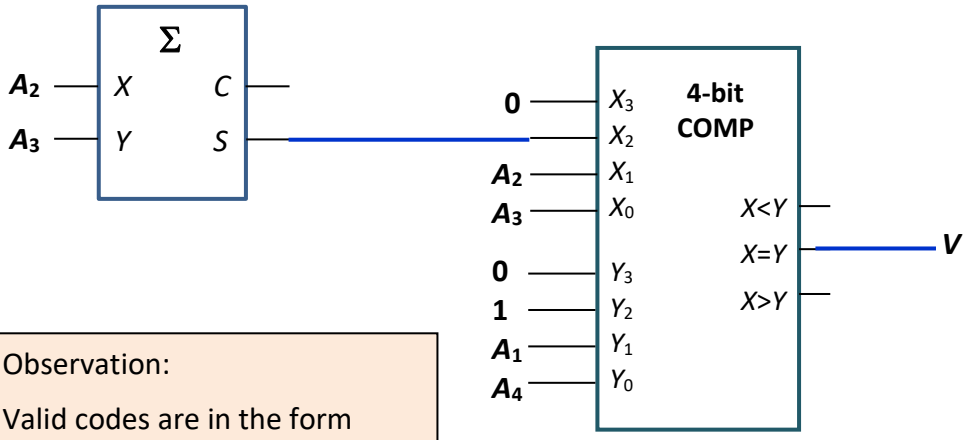
R = ~~*B·C'·D'*~~ ***B·C'·D'+ A·B·C'+ A·B·D'***

Q5: /12

6a.
[3]


Σ
P5
Q5
X
Y
C
S
Σ
P4
Q4
X
Y
C
S
Σ
P3
Q3
X
Y
C
S
Σ
P2
Q2
X
Y
C
S
Σ
P1
Q1
X
Y
C
S
Σ
P0
Q0
X
Y
C
S
Count-1
5
4
3
2
1
0
F2
F1
F0
D2
D1
D0

6b.
[5]



Observation:

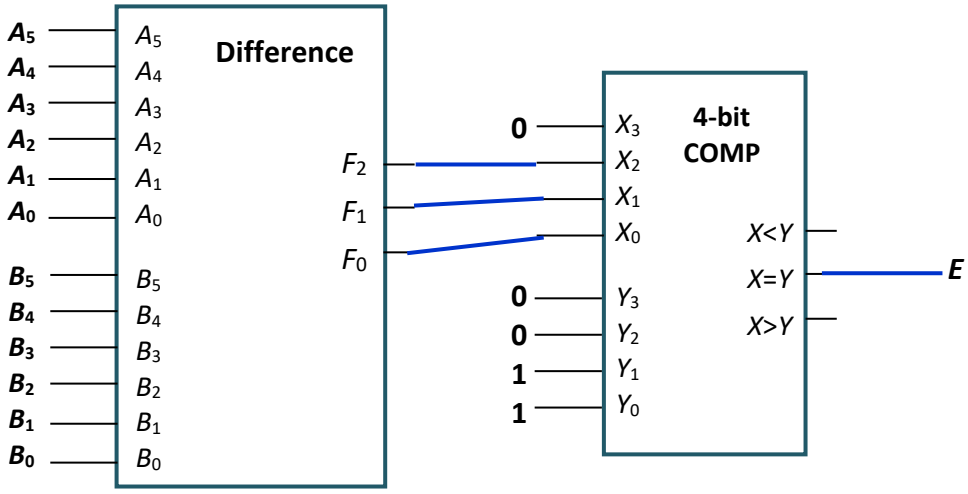
Valid codes are in the form

$A_4A_3A_2A_1$ = 0011 or 1100.

That is, $A_1 = A_2$, $A_3 = A_4$, $A_1 \neq A_3$.

See alternative answer on page 7.

6c.
[4]



Observation:

E=1 when there is a pair-wise bit difference of 3 in the two selected code values.

Q6: /12

7a. [1] **27**

b. [3] **26**

c. [3] **22**

d. [3] **3**

For (b)-(d): Partial credit for answers off by one, or answers in total number of cycles.

7e. [2]

Move Inst2 (add $t2, $0, $0) to the branch delay slot of Inst5 (beq $t0 $0, E1).

Not possible to find an instruction to fill in the branch-delay slot of Inst 7 (beq $t0, $0, E2) due to dependency.

7f. [2]

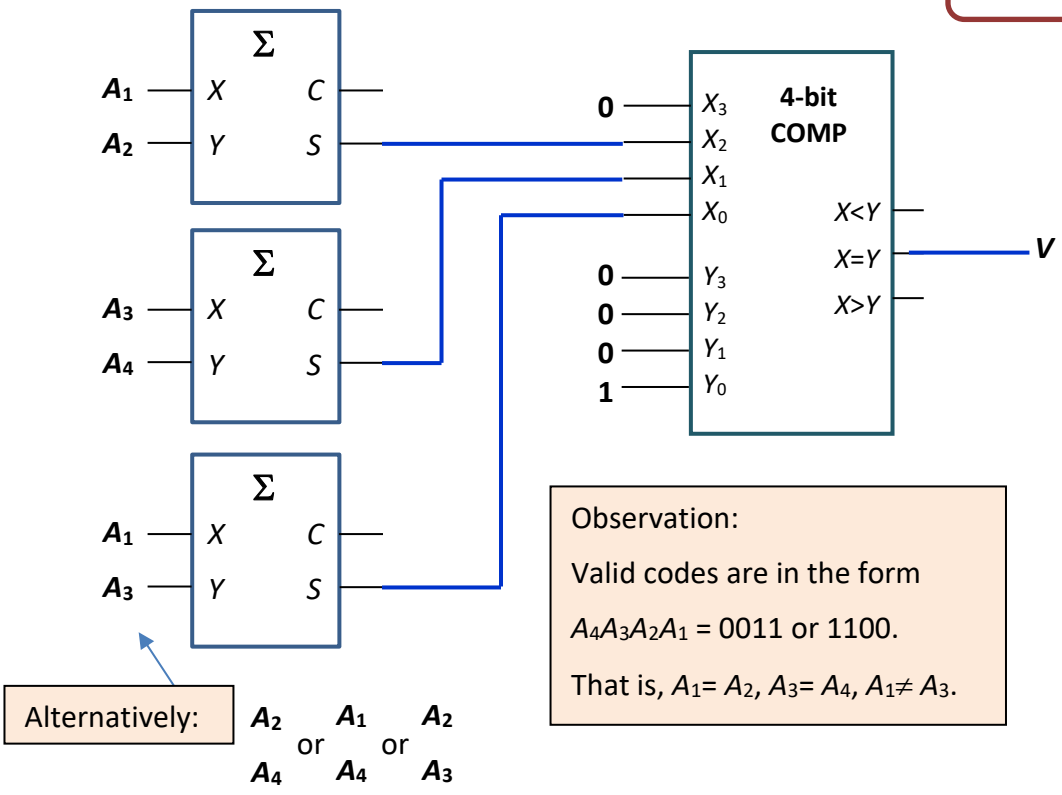
Predict not taken is easier to implement, as PC+4 is already computed. To implement predict taken, we need to compute the target branch address quickly in the first cycle. That requires additional hardware.

(In practice, we will need branch target prediction to predict the target address. This involves the use of Branch Target Buffer (BTB), which is usually cached. This is not covered in this module, so we accept any reasonable explanation.)

(It is wrong to say predict not taken is easier because then we predict correctly more often; this is not what the question asks for.)

Alternative answer for Q6(b): 4 marks.

Q7: /14



8a. [2] Index: 4 bits; Offset: 4 bits

8b. [2] *A*[0] → Block 12; *B*[60] → Block 11.

8c. [2] Hit rate for array *A*: 0; array *B*: 0.

8d. [2] *A*[0] → Set 4; *B*[60] → Set 3;

8e. [2] Hit rate for array *A*: 3/4; array *B*: 3/4.

8f. [2] **There is no change in hit rate compared to part (e) because the addresses of the accessed elements remain the same as in part (e). Therefore, hit rate is still 3/4.**

8g. [2]

	Word0	Word1	Word2	Word3
Index 0				
Index 1				
Index 2				
Index 3			**Inst1**	

8h. [3] Number of misses in the first iteration: 7

8i. [3] Number of misses in the second iteration: 4

Q8: /20

=== END OF PAPER ===

Workings

Q1 The code adds each hexadecimal digits in $s0 into $s1.

(a) 0 + C + 0 + C + E + B + E + B = **4A**.

(b) Since $s1 is the sum of digits in $s0, if $s0 = **0xFFFFFFFF**, $s1 will contain the maximum value of **0x78**.

(d) srl $s0, $s0, 4

Opcode rs rt rd shamt funct
000000 00000 10000 10000 00100 000010
0000 0000 0001 0000 1000 0001 0000 0010
0 0 1 0 8 1 0 2

Wrong answer:
000000 10000 00000 10000 00100 000010
0000 0010 0000 0000 1000 0001 0000 0010
0 2 0 0 8 1 0 2

(e) bne $s0, $zero, L = bne $s0, $zero, -4

Opcode rs rt immed
000101 10000 00000 1111 1111 1111 1100
0001 0110 0000 0000 1111 1111 1111 1100
1 6 0 0 F F F C

Wrong answer:
000101 00000 10000 1111 1111 1111 1100
0001 0100 0001 0000 1111 1111 1111 1100
1 4 1 0 F F F C

Q2 $-1.875_{10} = -1.111_2$
Normalise: `1. 11100000…`
Mantissa: `111 0000 0000 0000 0000 0000`
Exponent: `0` → `127 = 0111 1111`
Sign bit: `1`
Combined: `1011 1111 1111 0000 0000 0000 0000 0000`
Hexadecimal: **`0xBFF00000`**

Q3 (a) Idea: 6 MSB = 000000
Logical right shift.
Check equal 0 (but since always positive due to srl, < 1).

```
srl  $t0, $s0, 26
slti $t0, $t0, 1
```

(b) Idea: Get bits 21-25
Logical left shift to remove opcode.
Logical right shift to remove rt, rd, shamt, funct and bring to position.

```
sll $t1, $s1, 6
srl $t1, $t1, 26
```

(c) Any if-statement equivalence.

Q4a.

Current state		Next state		Flip flop *A*		Flip flop *B*	
A	*B*	A^+	B^+	*JA*	*KA*	*JB*	*KB*
0	0	1	0	1	X	0	X
0	1	1	1	1	X	X	0
1	0	0	1	X	1	1	X
1	1	0	0	X	1	X	1

5(b)

AB	*CD*	*AB<CD*	*AB>CD*	(MUX) S_1S_0	(DEMUX) S_1S_0	*PQRS*	*PQRS*
00	00	0	0	00	00	A000	0000
00	01	1	0	10	01	0C00	0000
00	10	1	0	10	01	0C00	0100
00	11	1	0	10	01	0C00	0100
01	00	0	1	01	10	00B0	0010
01	01	0	0	00	00	A000	0000
01	10	1	0	10	01	0C00	0100
01	11	1	0	10	01	0C00	0100
10	00	0	1	01	10	00B0	0000
10	01	0	1	01	10	00B0	0000
10	10	0	0	00	00	A000	1000
10	11	1	0	10	01	0C00	0100
11	00	0	1	01	10	00B0	0010
11	01	0	~~0~~1	0001	~~00~~10	~~A000~~00B0	~~1000~~0010
11	10	0	~~0~~1	~~00~~01	~~00~~10	~~A000~~00B0	~~1000~~0010
11	11	0	0	00	00	A000	1000

P

AB \ *CD*	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	~~1~~0	1	~~1~~0
10	0	0	0	1

~~***P = A·B·D + A·C·D'***~~

P = A·B·C·D + A·B'·C·D'

Q

AB \ *CD*	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

Q = A'·C + B'·C·D

R

AB \ *CD*	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	~~0~~1	0	~~0~~1
10	0	0	0	0

~~***R = B·C'·D'***~~

R = B·C'·D'+ A·B·C'+ A·B·D'

Q7.

Delays are highlighted under the columns (b), (c), (d) for parts (b), (c), (d) respectively.

```
                                                    (b)     (c)     (d)
      add  $t1, $0,  $0       # Inst1 :
      add  $t2, $0,  $0       # Inst2 :
      add  $t3, $0,  $0       # Inst3 :

L1:   slt  $t0, $t1, $s0      # Inst4 :
      beq  $t0, $0,  E1       # Inst5 :  +2     +2      +1

L2:   slt  $t0, $t2, $s0      # Inst6 :  +3     +1
      beq  $t0, $0,  E2       # Inst7 :  +2     +2      +1

      add  $t4, $s1, $t3      # Inst8 :  +3     +1
      lw   $t5, 0($t4)        # Inst9 :  +2     +2
      add  $t6, $s2, $t3      # Inst10:
      lw   $t7, 0($t6)        # Inst11:  +2     +2

      # Inst12-19 to create image C from A and B
      xor  $t9, $t5, $t7      # Inst12:  +2     +2      +1
      andi $t5, $t5, 0xFF     # Inst13:
      andi $t7, $t7, 0xFF     # Inst14:
      add  $t0, $t5, $t7      # Inst15:  +2     +2
      srl  $t0, $t0, 1        # Inst16:  +2     +2
      srl  $t9, $t9, 8        # Inst17:
      sll  $t9, $t9, 8        # Inst18:  +2     +2
      or   $t9, $t9, $t0      # Inst19:  +2     +2

      add  $t8, $s3, $t3      # Inst20:
      sw   $t9, 0($t8)        # Inst21:  +2     +2

      addi $t3, $t3, 4        # Inst22:
      addi $t2, $t2, 1        # Inst23:
                                 Total    +26     +22     +3
```

Q8. (a) 64 words; 1 block = 4 words (16 bytes) → 16 blocks.
Index: **4 bits**; Offset: **4 bits**.

(b) *A*[0] at 0x0040CCC0
0x0040CCC0 → 00 … 1100 1100 0000 → **Block 12**

B[0] at 0x000002C0 → *B*[60] at 0x000003B0 (60×4 = 240 = 0xF0)
0x000003B0 → 00 … 0011 1011 0000 → **Block 11**

(c) Since *A*[0] and *B*[0] are mapped to the same block, there will be cache thrashing and hence hit rate = 0.

(d) 64 words; 1 block = 4 words = 16 bytes → 16 blocks. 2 blocks per set → 8 sets.
Set index: **3 bits**; Offset: **4 bits**.

0x0040CCC0 → 00 … 1100 1100 0000 → **Set 4**
0x000003B0 → 00 … 0011 1011 0000 → **Set 3**

(e) *A*[0] and *B*[0] are mapped to the same set, sitting in the two blocks of that set.
There are 4 words in each block, the first word is a miss, the other three are hits.
Therefore, hit rate = 3/4.

(f) There is no change in hit rate compared to part (e) because the addresses of the accessed elements remain the same as in part (e).
Therefore, hit rate = 3/4.

(g) 0x04FFFFF8 → 00 … 1111 1000 → block 3, word 2.
Inst1 at block 3, word 2.

(h) First iteration, misses at instructions 1, 3, 7, 11, 15, 19, 23 → **7 misses**.

	Word0	Word1	Word2	Word3
Index 0	3 (M) 19 (M)	4 20	5 21	6 22
Index 1	7 (M) 23 (M)	8 24	9	10
Index 2	11 (M)	12	13	14
Index 3	15 (M)	16	1 (M) 17	2 18

(i) Second iteration, misses at instructions 6, 7, 19, 23 → **4 misses**.