CS2100

9 November 2021

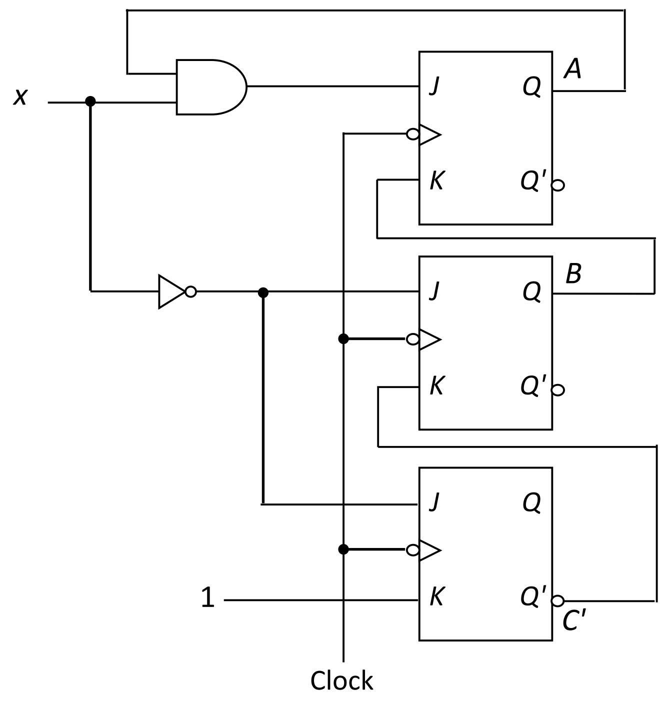
Selected Past Years' Exam Questions

(Prepared by: Aaron Tan)

Contents:

AY2020/21 Semester 2

AY2020/21 Semester 2 Q12 5 valid states: 000 to 100.



JA = *A*; *KA* = *B*

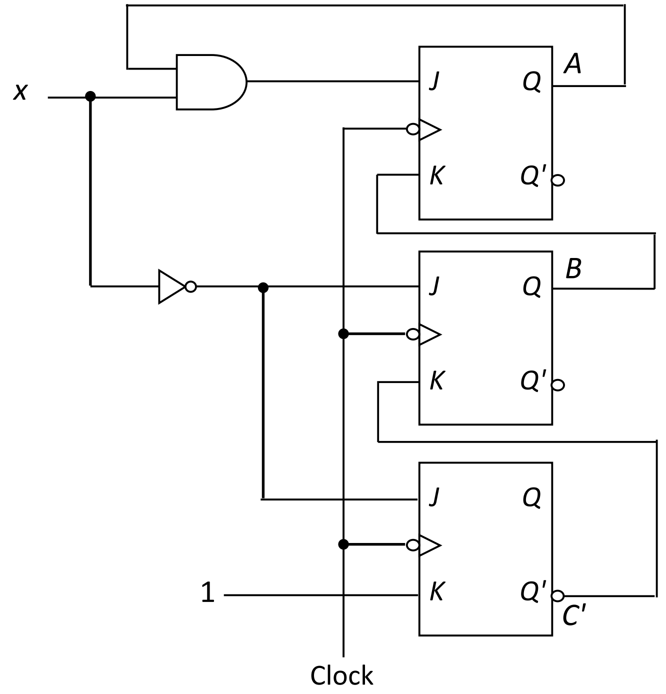
JB = *x*'; *KB* = *C*'

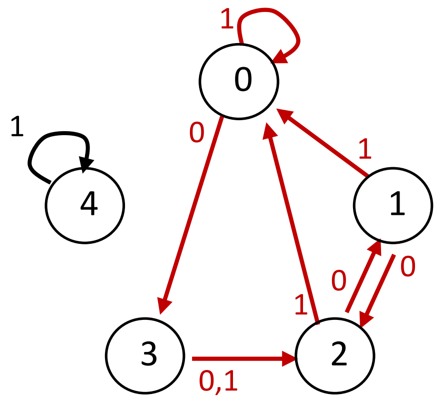
JC = *x*'; *KC* = 1

Present state			Input	Flip-flop A		Flip-flop B		Flip-flop C		Next state		
A	*B*	*C*	*x*	*JA*	*KA*	*JB*	*KB*	*JC*	*KC*	A^+	B^+	C^+
0	0	0	0									
0	0	0	1									
0	0	1	0									
0	0	1	1									
0	1	0	0									
0	1	0	1									
0	1	1	0									
0	1	1	1									
1	0	0	0									
1	0	0	1									
1	0	1	0									
1	0	1	1									
1	1	0	0									
1	1	0	1									
1	1	1	0									
1	1	1	1									

AY2020/21 Semester 2 Q12

5 valid states: 000 to 100.





Present state			Input	Flip-flop A		Flip-flop B		Flip-flop C		Next state		
A	B	C	x	JA	KA	JB	KB	JC	KC	A^+	B^+	C^+
0	0	0	0	0	0	1	1	1	1	0	1	1
0	0	0	1	0	0	0	1	0	1	0	0	0
0	0	1	0	0	0	1	0	1	1	0	1	0
0	0	1	1	0	0	0	0	0	1	0	0	0
0	1	0	0	0	1	1	1	1	1	0	0	1
0	1	0	1	0	1	0	1	0	1	0	0	0
0	1	1	0	0	1	1	0	1	1	0	1	0
0	1	1	1	0	1	0	0	0	1	0	1	0
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	0	0	1	0	1	1	0	0
1	0	1	0	0	0	1	0	1	1	1	1	0
1	0	1	1	1	0	0	0	0	1	1	0	0
1	1	0	0	0	1	1	1	1	1	0	0	1
1	1	0	1	1	1	0	1	0	1	0	0	0
1	1	1	0	0	1	1	0	1	1	0	1	0
1	1	1	1	1	1	0	0	0	1	0	1	0

AY2020/21 Semester 2 Q12 Remove state 4.

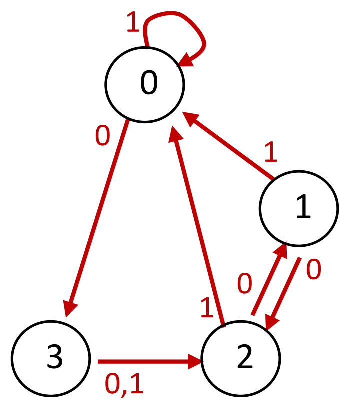
$$DF \qquad \begin{array}{c|c|c|c|c|} \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{2}{c}{G} \\ \cline{2-5} & 1 & 0 & 0 & 1 \\ \cline{2-5} F & 0 & 0 & 1 & 1 \\ \cline{2-5} \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{2}{c}{x} & \multicolumn{1}{c}{} \end{array}$$

$$DG \qquad \begin{array}{c|c|c|c|c|} \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{2}{c}{G} \\ \cline{2-5} & 1 & 0 & 0 & 0 \\ \cline{2-5} F & 1 & 0 & 0 & 0 \\ \cline{2-5} \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{2}{c}{x} & \multicolumn{1}{c}{} \end{array}$$

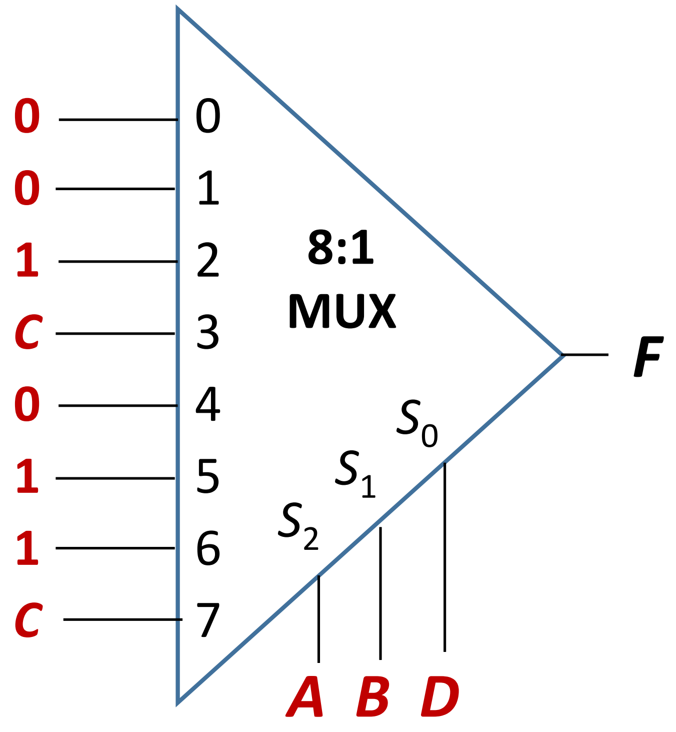
$DF = F' \cdot x' + F \cdot G$

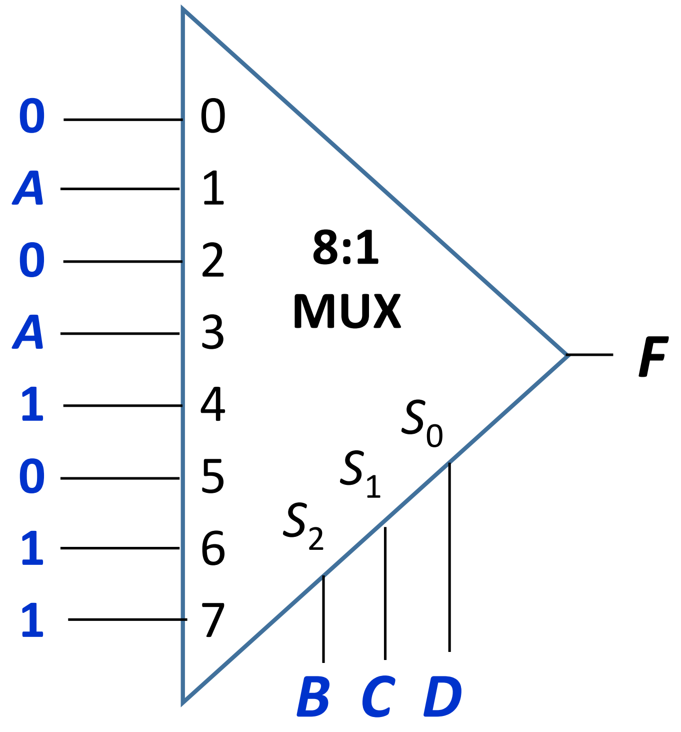
$DG = G' \cdot x'$

Present state		Input	Next state	
F	G	x	$DF = F^+$	$DG = G^+$
0	0	0	1	1
0	0	1	0	0
0	1	0	1	0
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	0



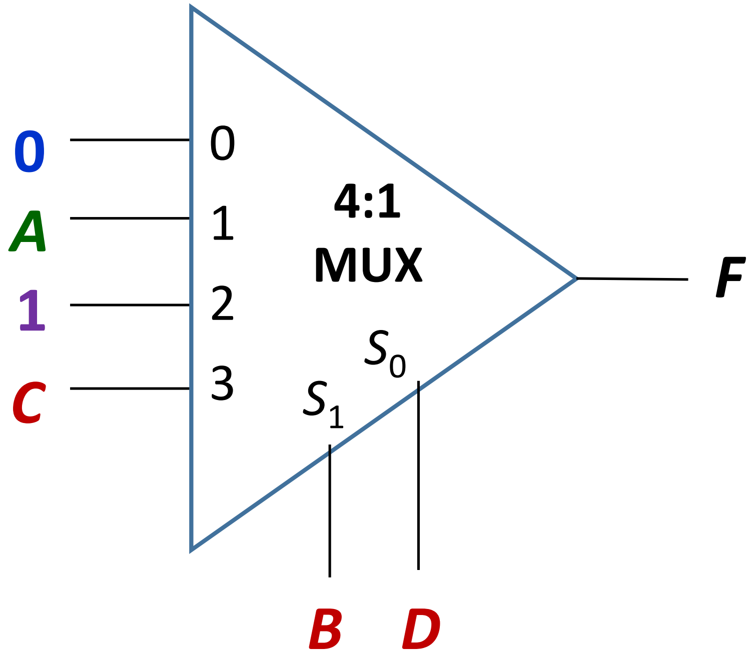
AY2020/21 Semester 2 Q13 (a) $F(A,B,C,D) = \Sigma m(4, 6, 7, 9, 11, 12, 14, 15)$.





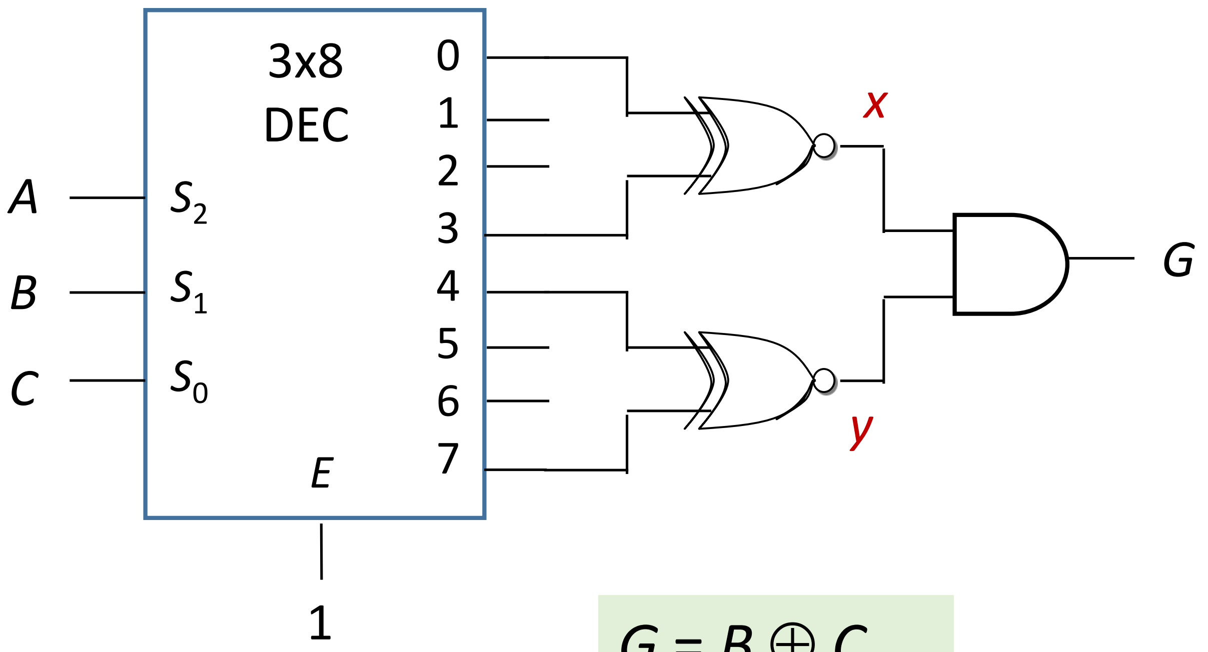
A	*B*	*C*	*D*	*F*
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

AY2020/21 Semester 2 Q13 (b) $F(A,B,C,D) = \Sigma m(4, 6, 7, 9, 11, 12, 14, 15)$.



A	B	C	D	F
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

AY2020/21 Semester 2 Q13 (c) *G*(*A*,*B*,*C*)



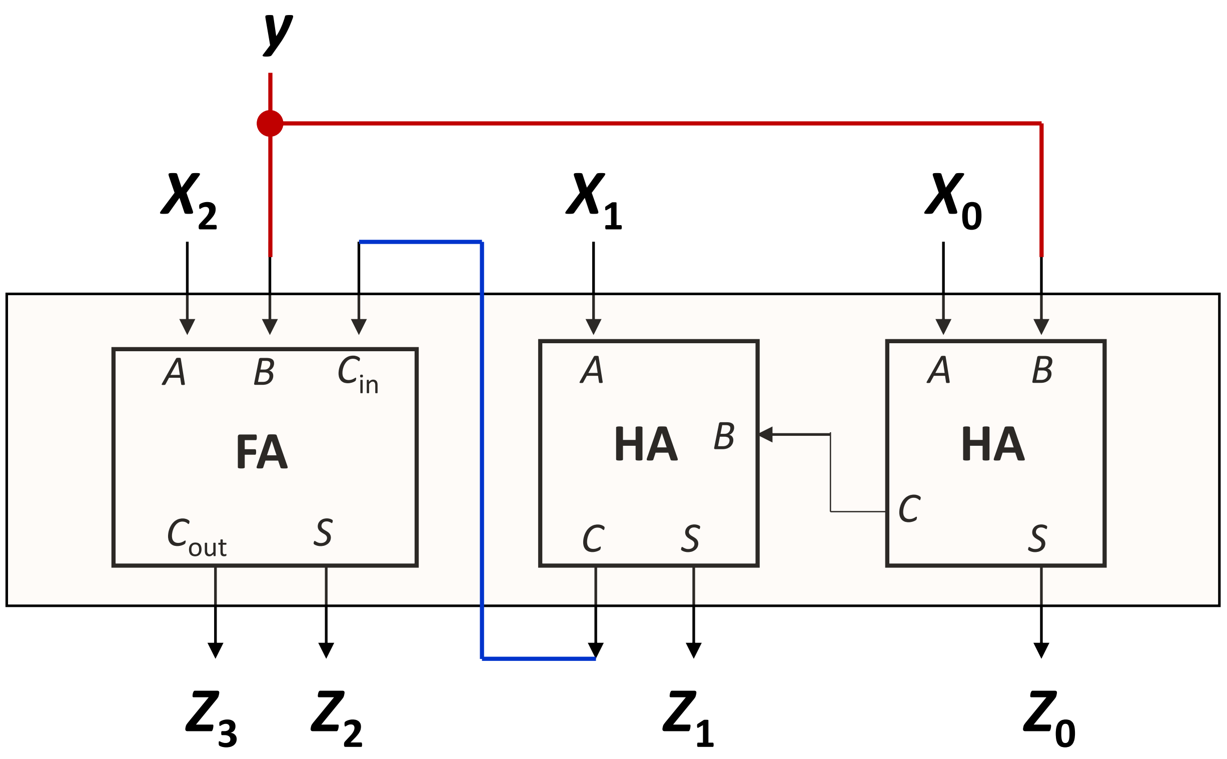
A	*B*	*C*	*x*	*y*	*G*
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

$G = B \oplus C$

AY2020/21 Semester 2 Q13

(d) Design a circuit that takes in a 3-bit unsigned binary number $X=X_2X_1X_0$ and a one-bit value y to generate the output $Z_3Z_2Z_1Z_0$ which is a binary number representing the value $X+5y$.

F-block



X_2	X_1	X_0	y	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	1
0	0	1	0	0	0	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	0	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	0	1	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	0	0
1	0	0	1	1	0	0	1
1	0	1	0	0	1	0	1
1	0	1	1	1	0	1	0
1	1	0	0	0	1	1	0
1	1	0	1	1	0	1	1
1	1	1	0	0	1	1	1
1	1	1	1	1	1	0	0

AY2020/21 Semester 2 Q14

```
.data
A: .word 10,21,12,17,9,1,20,33 # after: 10,18,12,2,7,-1,20,22
B: .word 100,3,20,15,2,2,65,11
size: .word 8

.text
main: la   $s0, A          # $s0 is the base address of array A
      la   $s1, B          # $s1 is the base address of array B
                           # Box 1

# ---------------------------------------------------------------
      add  $s5, $0, $0     # I1
      add  $t0, $0, $0     # I2
loop: slt  $t8, $t0, $s2   # I3
      beq  $t8, $0, end    # I4
      sll  $t1, $t0, 2     # I5
      add  $t3, $t1, $s0   # I6
      lw   $s3, 0($t3)     # I7
      andi $t9, $s3, 1     # I8
      beq  $t9, $0, skip   # I9
      add  $t4, $t1, $s1   # I10
      lw   $s4, 0($t4)     # I11
      sub  $s3, $s3, $s4   # I12
      sw   $s3, 0($t3)     # I13
      addi $s5, $s5, 1     # I14
skip: addi $t0, $t0, 1     # I15
      j    loop            # I16
# ---------------------------------------------------------------
end:                       # Box 2


      syscall              # system call to print
      li   $v0, 10
      syscall              # system call to exit
```

- $s0 = base address of array *A*
- $s1 = base address of array *B*
- $s2 = size of array *A*
- $s5 = count

(a) Fill in Box 1 with MIPS instruction(s) to load the value of *size* into $s2.

```
la $t0, size
lw $s2, 0($t0)
```

(b) Fill in Box 1 with MIPS instruction(s) to prepare for the printing of the value of *count* ($s5).

```
li  $v0, 1
add $a0, $s5, $0
```

AY2020/21 Semester 2 Q14

```
.data
A: .word 10,21,12,17,9,1,20,33 # after: 10,18,12,2,7,-1,20,22
B: .word 100,3,20,15,2,2,65,11
size: .word 8

.text
main: la   $s0, A          # $s0 is the base address of array A
      la   $s1, B          # $s1 is the base address of array B
      [              ]     # Box 1

# ---------------------------------------------------------------
      add  $s5, $0, $0     # I1
      add  $t0, $0, $0     # I2
loop: slt  $t8, $t0, $s2   # I3
      beq  $t8, $0, end    # I4
      sll  $t1, $t0, 2     # I5
      add  $t3, $t1, $s0   # I6
      lw   $s3, 0($t3)     # I7
      andi $t9, $s3, 1     # I8
      beq  $t9, $0, skip   # I9
      add  $t4, $t1, $s1   # I10
      lw   $s4, 0($t4)     # I11
      sub  $s3, $s3, $s4   # I12
      sw   $s3, 0($t3)     # I13
      addi $s5, $s5, 1     # I14
skip: addi $t0, $t0, 1     # I15
      j    loop            # I16
# ---------------------------------------------------------------
end:  [              ]     # Box 2

      syscall              # system call to print
      li   $v0, 10
      syscall              # system call to exit
```

- $s0 = base address of array *A*
- $s1 = base address of array *B*
- $s2 = size of array *A*
- $s5 = count

(c) Write an equivalent C code for I1 to I16.

```
count = 0;
for (j=0; j<size; j++) {
   if (A[j]%2 == 1) {
      A[j] = A[j] - B[j];
      count = count + 1;
   }
}
```

AY2020/21 Semester 2 Q14

```
.data
A: .word 10,21,12,17,9,1,20,33 # after: 10,18,12,2,7,-1,20,22
B: .word 100,3,20,15,2,2,65,11
size: .word 8

.text
main: la   $s0, A          # $s0 is the base address of array A
      la   $s1, B          # $s1 is the base address of array B
                           # Box 1

# --------------------------------------------------------------
      add  $s5, $0, $0     # I1
      add  $t0, $0, $0     # I2
loop: slt  $t8, $t0, $s2   # I3
      beq  $t8, $0, end    # I4
      sll  $t1, $t0, 2     # I5
      add  $t3, $t1, $s0   # I6
      lw   $s3, 0($t3)     # I7
      andi $t9, $s3, 1     # I8
      beq  $t9, $0, skip   # I9
      add  $t4, $t1, $s1   # I10
      lw   $s4, 0($t4)     # I11
      sub  $s3, $s3, $s4   # I12
      sw   $s3, 0($t3)     # I13
      addi $s5, $s5, 1     # I14
skip: addi $t0, $t0, 1     # I15
      j    loop            # I16
# --------------------------------------------------------------
end:                       # Box 2

      syscall              # system call to print
      li   $v0, 10
      syscall              # system call to exit
```

- $s0 = base address of array *A*
- $s1 = base address of array *B*
- $s2 = size of array *A*
- $s5 = count

(d) Instruction encoding of I3.

`slt $t8, $t0, $s2`

opcode = 0; funct = 0x2A = 0b101010

rd = $t8 = $24 = 0b11000
rs = $t0 = $8 = 0b01000
rt = $s2 = $18 = 0b10010

opcode rs rt rd shamt funct
000000 01000 10010 11000 00000 101010

0x 0 1 1 2 C 0 2 A

AY2020/21 Semester 2 Q14

```
.data
A: .word 10,21,12,17,9,1,20,33 # after: 10,18,12,2,7,-1,20,22
B: .word 100,3,20,15,2,2,65,11
size: .word 8

.text
main: la   $s0, A          # $s0 is the base address of array A
      la   $s1, B          # $s1 is the base address of array B
      ┌──────────────────┐  # Box 1
      │                  │
      └──────────────────┘
# --------------------------------------------------------------
      add  $s5, $0, $0     # I1
      add  $t0, $0, $0     # I2
loop: slt  $t8, $t0, $s2   # I3
      beq  $t8, $0, end    # I4
      sll  $t1, $t0, 2     # I5
      add  $t3, $t1, $s0   # I6
      lw   $s3, 0($t3)     # I7
      andi $t9, $s3, 1     # I8
      beq  $t9, $0, skip   # I9
      add  $t4, $t1, $s1   # I10
      lw   $s4, 0($t4)     # I11
      sub  $s3, $s3, $s4   # I12
      sw   $s3, 0($t3)     # I13
      addi $s5, $s5, 1     # I14
skip: addi $t0, $t0, 1     # I15
      j    loop            # I16
# --------------------------------------------------------------
end:  ┌──────────────────┐  # Box 2
      │                  │
      └──────────────────┘
      syscall              # system call to print
      li   $v0, 10
      syscall              # system call to exit
```

- $s0 = base address of array *A*
- $s1 = base address of array *B*
- $s2 = size of array *A*
- $s5 = count

(e) Instruction encoding of I9.

`beq $t9, $0, skip`

opcode = 0x4 = 0b000100

rs = $t9 = $25 = 0b11001
rt = $0 = 0b00000
skip = 5

opcode rs rt immed
000100 11001 00000 0000000000000101

`0x 1 3 2 0 0 0 0 5`

AY2020/21 Semester 2 Q14

```
.data
A: .word 10,21,12,17,9,1,20,33 # after: 10,18,12,2,7,-1,20,22
B: .word 100,3,20,15,2,2,65,11
size: .word 8

.text
main: la   $s0, A          # $s0 is the base address of array A
      la   $s1, B          # $s1 is the base address of array B
                           # Box 1

# ----------------------------------------------------------------
      add  $s5, $0, $0     # I1
      add  $t0, $0, $0     # I2
loop: slt  $t8, $t0, $s2   # I3
      beq  $t8, $0, end    # I4
      sll  $t1, $t0, 2     # I5
      add  $t3, $t1, $s0   # I6
      lw   $s3, 0($t3)     # I7
      andi $t9, $s3, 1     # I8
      beq  $t9, $0, skip   # I9
      add  $t4, $t1, $s1   # I10
      lw   $s4, 0($t4)     # I11
      sub  $s3, $s3, $s4   # I12
      sw   $s3, 0($t3)     # I13
      addi $s5, $s5, 1     # I14
skip: addi $t0, $t0, 1     # I15
      j    loop            # I16
# ----------------------------------------------------------------
end:                       # Box 2

      syscall              # system call to print
      li   $v0, 10
      syscall              # system call to exit
```

- \$s0 = base address of array *A*
- \$s1 = base address of array *B*
- \$s2 = size of array *A*
- \$s5 = count

(f) Instruction encoding of I16. I1 at address 0x10000C50.

`j loop`

opcode = 0x2 = 0b000010

Address at loop:
0x10000C58
0b ~~0001~~ 0000 0000 0000 0000 1100 0101 10~~00~~

opcode immed
0b 000010 0000 0000 0000 0000 1100 0101 10

0x 0 8 0 0 0 3 1 6

AY2020/21 Semester 2 Q15

```
      add  $s5, $0, $0    # I1
      add  $t0, $0, $0    # I2
loop: slt  $t8, $t0, $s2  # I3
      beq  $t8, $0, end   # I4
      sll  $t1, $t0, 2    # I5
      add  $t3, $t1, $s0  # I6
      lw   $s3, 0($t3)    # I7
      andi $t9, $s3, 1    # I8
      beq  $t9, $0, skip  # I9
      add  $t4, $t1, $s1  # I10
      lw   $s4, 0($t4)    # I11
      sub  $s3, $s3, $s4  # I12
      sw   $s3, 0($t3)    # I13
      addi $s5, $s5, 1    # I14
skip: addi $t0, $t0, 1    # I15
      j    loop           # I16
```

Direct-mapped data cache with 128 words in total and each block contains 4 words.

- Data cache used for lw, not for sw.
- Array *A* starts at 0xFF000C00; all elements +ve odd integers.
- Array *B* follows array *A* immediately.

(a) How many bits in the index field? In the byte offset field?

128/4 = 32 blocks → 5 bits for index field.

16 bytes per block → 4 bits for byte offset field.

AY2020/21 Semester 2 Q15

```
      add  $s5, $0, $0    # I1
      add  $t0, $0, $0    # I2
loop: slt  $t8, $t0, $s2  # I3
      beq  $t8, $0, end   # I4
      sll  $t1, $t0, 2    # I5
      add  $t3, $t1, $s0  # I6
      lw   $s3, 0($t3)    # I7
      andi $t9, $s3, 1    # I8
      beq  $t9, $0, skip  # I9
      add  $t4, $t1, $s1  # I10
      lw   $s4, 0($t4)    # I11
      sub  $s3, $s3, $s4  # I12
      sw   $s3, 0($t3)    # I13
      addi $s5, $s5, 1    # I14
skip: addi $t0, $t0, 1    # I15
      j    loop           # I16
```

Direct-mapped data cache with 128 words in total and each block contains 4 words.

- Data cache used for lw, not for sw.
- Array *A* starts at 0xFF000C00; all elements +ve odd integers.
- Array *B* follows array *A* immediately.

(b) Array A contains 3200 elements. What is the hit rate for array *A*? for array *B*?

Array *A* starts at 0xFF000C00 = 0b…110 00000 0000.

Array *B* starts at 0xFF000C00 + (3200×4) = 0xFF003E00
= 0b…111 00000 0000.

A[0] and *B*[0] loaded into block 0 word 0. In fact, *A*[i] and *B*[i] (0 < i < 3199) are loaded into the same block same word → cache thrashing.

Therefore, 0 hit rate for arrays *A* and *B*.

AY2020/21 Semester 2 Q15

```
      add  $s5, $0, $0    # I1
      add  $t0, $0, $0    # I2
loop: slt  $t8, $t0, $s2  # I3
      beq  $t8, $0, end   # I4
      sll  $t1, $t0, 2    # I5
      add  $t3, $t1, $s0  # I6
      lw   $s3, 0($t3)    # I7
      andi $t9, $s3, 1    # I8
      beq  $t9, $0, skip  # I9
      add  $t4, $t1, $s1  # I10
      lw   $s4, 0($t4)    # I11
      sub  $s3, $s3, $s4  # I12
      sw   $s3, 0($t3)    # I13
      addi $s5, $s5, 1    # I14
skip: addi $t0, $t0, 1    # I15
      j    loop           # I16
```

Direct-mapped data cache with 128 words in total and each block contains 4 words.

- Data cache used for lw, not for sw.
- Array *A* starts at 0xFF000C00; all elements +ve odd integers.
- Array *B* follows array *A* immediately.

(c) Array A contains 3216 elements. What is the hit rate for array *A*? for array *B*?

Array *A* starts at 0xFF000C00 = 0b…110 00000 0000.

Array *B* starts at 0xFF000C00 + (3216×4) = 0xFF003E40 = 0b…111 00100 0000.

A[0] loaded into block 0 word 0. *B*[0] loaded into block 4 word 0.

First 4 iterations *A*[0] is miss and *A*[1] to *A*[3] are hits. Same for array *B*.

Therefore, 75% hit rate for arrays *A* and *B*.

AY2020/21 Semester 2 Q15

```
      add  $s5, $0, $0    # I1
      add  $t0, $0, $0    # I2
loop: slt  $t8, $t0, $s2  # I3
      beq  $t8, $0, end   # I4
      sll  $t1, $t0, 2    # I5
      add  $t3, $t1, $s0  # I6
      lw   $s3, 0($t3)    # I7
      andi $t9, $s3, 1    # I8
      beq  $t9, $0, skip  # I9
      add  $t4, $t1, $s1  # I10
      lw   $s4, 0($t4)    # I11
      sub  $s3, $s3, $s4  # I12
      sw   $s3, 0($t3)    # I13
      addi $s5, $s5, 1    # I14
skip: addi $t0, $t0, 1    # I15
      j    loop           # I16
```

(d) Array A contains 3210 elements. What is the hit rate for array *A*? for array *B*?

Array *A* starts at 0xFF000C00 = 0b…110 00000 0000.

Array *B* starts at 0xFF000C00 + (3216×4) = 0xFF003E28
= 0b…111 00010 1000.

A[0] loaded into block 0 word 0. *B*[0] loaded into block 2 word 2.

First 8 iterations:

Index 0	A[0]	A[1]	A[2]	A[3]
Index 1	A[4]	A[5]	A[6]	A[7]
Index 2	…	…	B[0]	B[1]
Index 3	B[2]	B[3]	B[4]	B[5]
Index 4	B[6]	B[7]	…	…

:

Cache access pattern for array *A*: (M,H,H,H),…,M,H.
The (M,H,H,H) is repeated 802 times.

Therefore, hit rate for array *A* is 2407/3210 or 74.98%.

Cache access pattern for array *B*: M,H,(M,H,H,H),….
The (M,H,H,H) is repeated 802 times.

Therefore, hit rate for array *B* is the same as array *A*.

AY2020/21 Semester 2 Q15

```
      add  $s5, $0, $0    # I1
      add  $t0, $0, $0    # I2
loop: slt  $t8, $t0, $s2  # I3
      beq  $t8, $0, end   # I4
      sll  $t1, $t0, 2    # I5
      add  $t3, $t1, $s0  # I6
      lw   $s3, 0($t3)    # I7
      andi $t9, $s3, 1    # I8
      beq  $t9, $0, skip  # I9
      add  $t4, $t1, $s1  # I10
      lw   $s4, 0($t4)    # I11
      sub  $s3, $s3, $s4  # I12
      sw   $s3, 0($t3)    # I13
      addi $s5, $s5, 1    # I14
skip: addi $t0, $t0, 1    # I15
      j    loop           # I16
```

2-way set-associative instruction cache with 16 words in total and each block contains 2 words.

- 100 elements in array *A*, all positive odd integers.
- 100 elements in array *B*.

(e) How many bits in the set index field? In the byte offset field?

16/2/2 = 4 sets → 2 bits for set index field.

8 bytes per block → 3 bits for byte offset field.

AY2020/21 Semester 2 Q15

```
      add  $s5, $0, $0    # I1
      add  $t0, $0, $0    # I2
loop: slt  $t8, $t0, $s2  # I3
      beq  $t8, $0, end   # I4
      sll  $t1, $t0, 2    # I5
      add  $t3, $t1, $s0  # I6
      lw   $s3, 0($t3)    # I7
      andi $t9, $s3, 1    # I8
      beq  $t9, $0, skip  # I9
      add  $t4, $t1, $s1  # I10
      lw   $s4, 0($t4)    # I11
      sub  $s3, $s3, $s4  # I12
      sw   $s3, 0($t3)    # I13
      addi $s5, $s5, 1    # I14
skip: addi $t0, $t0, 1    # I15
      j    loop           # I16
```

2-way set-associative instruction cache with 16 words in total and each block contains 2 words.

- 100 elements in array *A*, all positive odd integers.
- 100 elements in array *B*.

(f) I1at 0x10000C50. How many hits in the execution of the code?

0x10000C50 = 0b00… 010 10 000

I1 is loaded into set 2 word 0.

After the first iteration:

Set 0	I5	I6	I13	I14
Set 1	I7	I8	I15	I16
Set2	I1	I2	I9	I10
Set 3	I3	I4	I11	I12

First iteration: 8 hits.

Subsequent iterations (I3 – I16): 14 hits.

After loop (I3, I4): 2 hits.

Therefore, total hits = 8 + (99×14) + 2 = **1396**.

AY2020/21 Semester 2 Q15

```
      add  $s5, $0, $0    # I1
      add  $t0, $0, $0    # I2
loop: slt  $t8, $t0, $s2  # I3
      beq  $t8, $0, end   # I4
      sll  $t1, $t0, 2    # I5
      add  $t3, $t1, $s0  # I6
      lw   $s3, 0($t3)    # I7
      andi $t9, $s3, 1    # I8
      beq  $t9, $0, skip  # I9
      add  $t4, $t1, $s1  # I10
      lw   $s4, 0($t4)    # I11
      sub  $s3, $s3, $s4  # I12
      sw   $s3, 0($t3)    # I13
      addi $s5, $s5, 1    # I14
skip: addi $t0, $t0, 1    # I15
      j    loop           # I16
```

2-way set-associative instruction cache with 16 words in total and each block contains 2 words.

- 100 elements in array *A*, all positive odd integers.
- 100 elements in array *B*.

(g) I1at 0x10000C54. How many hits in the execution of the code?

0x10000C54 = 0b00… 010 10 100

I1 is loaded into set 2 word 1.

After the first iteration:

Set 0	I4	I5	I12	I13
Set 1	I6	I7	I14	I15
Set2	I16	~~I1~~	I8	I9
Set 3	I2	I3	I10	I11

First iteration: 7 hits.

Subsequent iterations (I3 – I16): 14 hits.

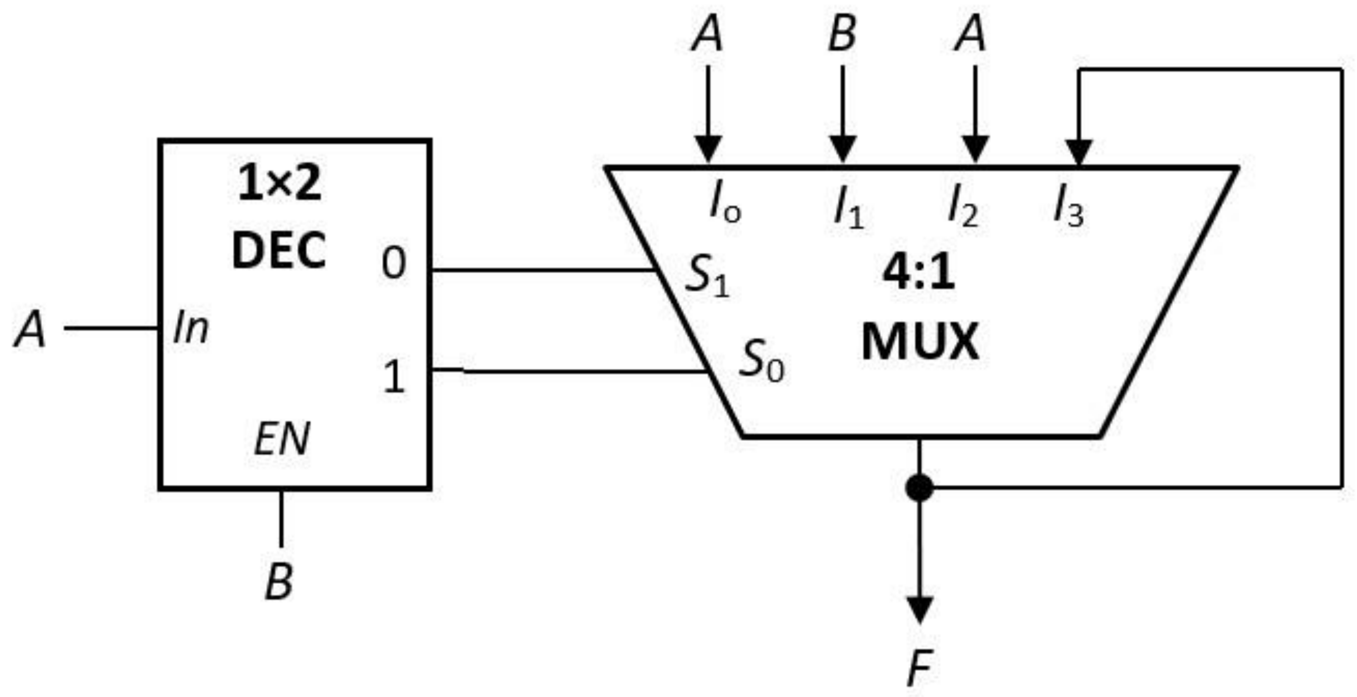
After loop (I3, I4): 2 hits.

Therefore, total hits = 7 + (99×14) + 2 = **1395**.

AY2017/18 Semester 2

AY2017/18 Semester 2 Q3

(a) Given the following circuit, what is ***F***?



A	***B***	***F***
0	0	0
0	1	0
1	0	1
1	1	1

$F = A$

AY2017/18 Semester 2 Q4

(a) Suppose MIPS instructions in R-format must use the following 5 opcodes (in decimal): 0, 1, 16, 17 and 32, what is the maximum total number of instructions that can be supported in MIPS?

Answer: **379**

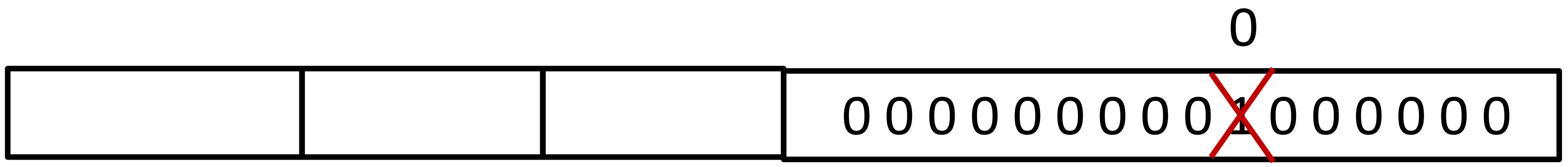
	6-bit		6-bit	
	opcode		funct	
Non-R-format:	(2^6-5)			$= 59$
R-format:	5	$\times$	2^6	$= 320$

Total: $59 + 320 = 379$

AY2017/18 Semester 2 Q4

(b) Suppose due to a hardware defect in the datapath circuit, a **stuck-at-0 fault** occurs at **bit 6** of every MIPS instruction. This means that bit 6 of a MIPS instruction is always 0 regardless of what the instruction is originally. Devise a simple test using a MIPS instruction to discover this error. Explain your test.

One possible test: addi $t0, $zero, 64



This is supposed to put 64 into $t0.

However, due to stuck-at-0 fault in bit 6, the final value in $t0 will be 0.

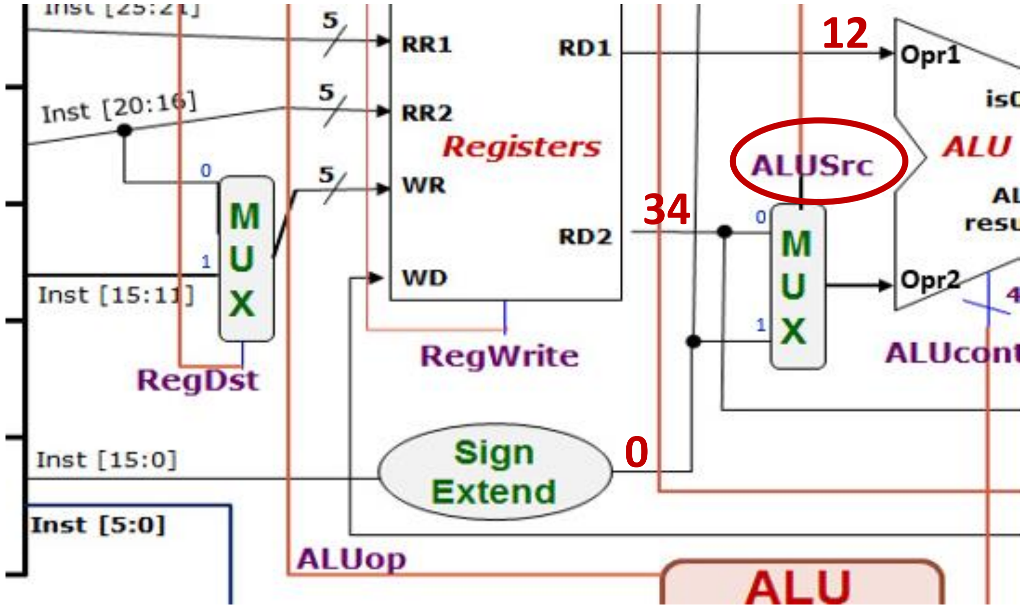
Other appropriate examples are accepted.

AY2017/18 Semester 2 Q4

(c) Suppose the **stuck-at-0 fault** occurs at the **ALUSrc** control signal.

Assuming that \$t0 and \$t1 contain 12 and 34 respectively, and we are to use the instruction **lw \$t1, 0(\$t0)** to discover the error. Describe what other preparation work needs to be done. You may assume that we can write data into any location in the memory.

Note that **ALUSrc** chooses between RD2 and SignExt(Inst[15:0]).



If the instruction is correctly carried out, **lw \$t1, 0(\$t0)** would have loaded the value at memory address 12 into \$t1.

With stuck-at-0 fault at ALUSrc, the instruction would have loaded the value at memory address 46 into \$t1 instead.

Hence, we could first load different values in addresses 12 and 46 before using the test.

AY2017/18 Semester 2 Q4

(d) You want to add the **bne** instruction into the datapath, which already includes the required hardware for the instruction. Write out the **ALUop** for **bne** and how you can determine whether the **bne** results in the branch to be taken.

Opcode	ALUop
lw	00
sw	00
beq	01
R-type	10
:	:

Since ALUop code 11 is not used, we may use it for **bne**.

Hence, branch taken = **ALUop1** AND **ALUop2** AND !(**isZero**) where (isZero) is the output from the ALU.

or

Use ALUop code 01 (same as **beq**).

Hence, branch taken = !**ALUop1** AND **ALUop2** AND !(**isZero**) where (isZero) is the output from the ALU, or

branch taken = **Branch** AND !(**isZero**) .

AY2017/18 Semester 2 Q7

$s0: base addr. of array *A*;
$s1: base addr. of array *B*.

$s2: *n*, #elements in array *A* (and *B*)

```
        beq  $s2, $zero, End   # Inst1, Addr: 0x0040003c
        addi $t8, $s2, -1      # Inst2
        sll  $t8, $t8, 2       # Inst3
Loop:   add  $t0, $s0, $t8     # Inst4
        add  $t1, $s1, $t8     # Inst5
        lw   $t2, 0($t0)       # Inst6
        lw   $t3, 0($t1)       # Inst7
        andi $t4, $t3, 3       # Inst8
        addi $t4, $t4, -3      # Inst9
        beq  $t4, $zero, A1    # Inst10
        add  $t2, $t2, $t3     # Inst11
        j    A2                # Inst12
A1:     addi $t2, $t2, 1       # Inst13
A2:     sw   $t2, 0($t0)       # Inst14
        addi $t8, $t8, -8      # Inst15
        slt  $t7, $t8, $zero   # Inst16
        beq  $t7, $zero, Loop  # Inst17
End:
```

```
int i;
for (i=n-1; i>=0; i-=2) {
  if (B[i]%4 == 3)
     A[i]++;
  else
     A[i] += B[i];
}
```

AY2017/18 Semester 2 Q7

Assuming that arrays *A* and *B* now each contains 1024 positive integers. Given a **direct-mapped data cache** with 128 words in total, each block containing 4 words with each word being 4 bytes long, arrays *A* and *B* are stored starting at memory addresses **0x10001000 and 0x1003F100** respectively.

The data cache is involved when memory is accessed (i.e., when **lw** and **sw** are executed.)

(a) How many bits are there in the index field? In the byte offset field?

Number of blocks = 128/4 = 32 = 2^5 → index field: 5 bits

Number of bytes per block = 16 = 2^4 → byte offset field: 4 bits

(b) Which index is *A*[1023] mapped to? Which index is *B*[1023] mapped to?

A[0] at 0x10001000 → *A*[1024] at 0x10002000 → *A*[1023] at 0x10001FFC

10001FFC → … 111$\underline{1\ 1111}$ 1100 → Index **31**

B[0] at 0x1003F100 → *B*[1024] at 0x10040100 → *B*[1023] at 0x100400FC

100400FC → … 000$\underline{0\ 1111}$ 1100 → Index **15**

AY2017/18 Semester 2 Q7

(c) How many memory accesses in total are made for array *A*? For array *B*?

512 elements for each array are accessed:
[1023], [1021], …, [3], [1].

A: **1024** accesses (1 read and 1 write per element);
B: **512** accesses.

```
int i;
for (i=n-1; i>=0; i-=2) {
   if (B[i]%4 == 3)
      A[i]++;
   else
      A[i] += B[i];
}
```

(d) What is the cache hit rate for array *A*? For array *B*?

Since *A*[1023] and *B*[1023] are mapped to different indices, there will not be any cache thrashing.

A: 256 misses, 768 hits, hit rate = **75%**.
B: 256 misses, 256 hits, hit rate = **50%**.

Index				
15	*B*[1020]	*B*[1021]	*B*[1022]	*B*[1023]
31	*A*[1020]	*A*[1021]	*A*[1022]	*A*[1023]

AY2017/18 Semester 2 Q7

(e) Given a **direct-mapped instruction cache** with 16 words in total, each block containing 2 instructions (words), and the first **beq** instruction is at memory address **0x0040003c**. How many cache hits and misses are there in total during the execution of the code, assuming that the **beq** instruction at Inst10 always branches to *A1*? You may consider only the instructions in the given code segment, i.e., Inst1 – Inst17.

Total #instructions = 3 + 512×12 = 6147.

Cache index field: 3 bits; byte offset: 3 bits.

First instruction at 0x0040003c →
… 0011 1100 → index 7, word 1 (second word)

Cache content in first iteration:
Misses: I1, I2, I4, I6, I8, I10, I13, I14 and I16.

Subsequently, all instructions are in the cache.

Therefore, **9 misses, 6138 hits**.

Index			
0	I2	I3	
1	I4	I5	
2	I6	I7	
3	I8	I9	
4	I10		
5		I13	
6	I14	I15	
7	I16	~~I1~~	I17

AY2016/17 Semester 2

AY2016/17 Semester 2 Midterm Q4

Simplify the following expression of a 15-variable Boolean function.

m's are the minterms and *M*'s the maxterms.

$$\underbrace{(m3125 \cdot M987 \cdot m1025)}_{0} + m895 \cdot \underbrace{(M2222 + M618)}_{1}$$

Answer: ***m895***

If $\alpha \neq \beta$,

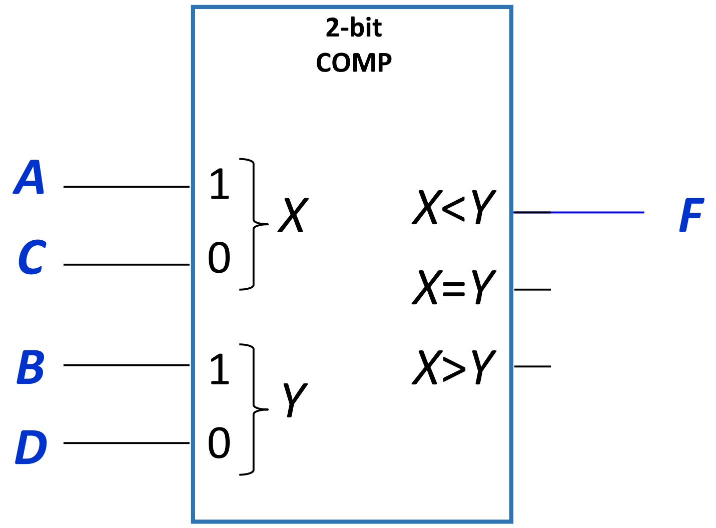
$$m\alpha \cdot m\beta = 0$$

$$M\alpha + M\beta = 1$$

AY2016/17 Semester 2 Exam Q2

(a) $F(A,B,C,D) = \Sigma m(1,4,5,6,7,13)$

Implement using a 2-bit magnitude comparator with no logic gates.



A	B	C	D	A	C	B	D	F=(AC<BD)
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1	1
0	0	1	0	0	1	0	0	0
0	0	1	1	0	1	0	1	0
0	1	0	0	0	0	1	0	1
0	1	0	1	0	0	1	1	1
0	1	1	0	0	1	1	0	1
0	1	1	1	0	1	1	1	1
1	0	0	0	1	0	0	0	0
1	0	0	1	1	0	0	1	0
1	0	1	0	1	1	0	0	0
1	0	1	1	1	1	0	1	0
1	1	0	0	1	0	1	0	0
1	1	0	1	1	0	1	1	1
1	1	1	0	1	1	1	0	0
1	1	1	1	1	1	1	1	0

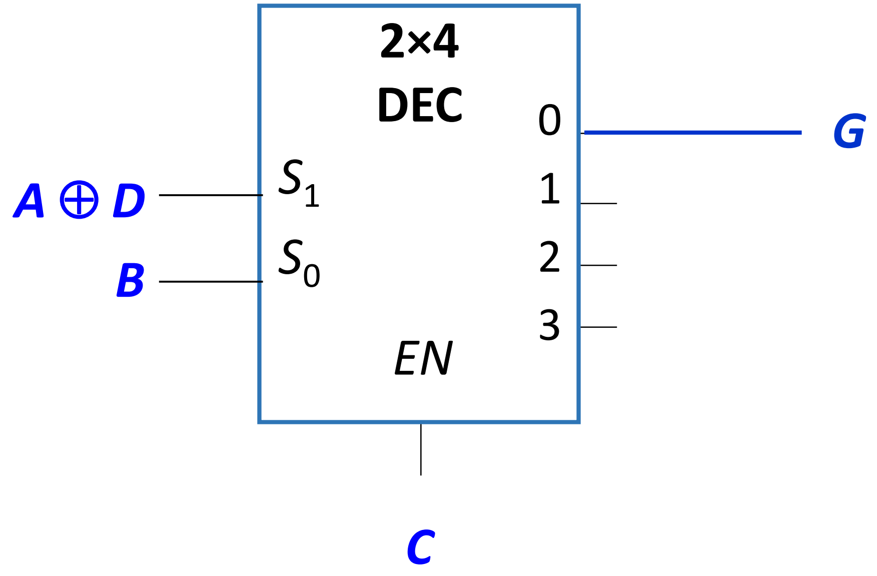
AY2016/17 Semester 2 Exam Q2

(b) $G(A,B,C,D) = \Sigma m(2,11)$

Implement using a 2×4 decoder and one XOR gate.

$$\overset{(A \oplus D)'}{}$$

$$\Sigma m(2,11) = A'\cdot B'\cdot C\cdot D' + A\cdot B'\cdot C\cdot D = C\cdot((A'\cdot D' + A\cdot D)\cdot B')$$



AY2016/17 Semester 2 Exam Q6

Cache access time = 5ns, main memory access time = 80ns.

(a) What is the memory access time when you have a cache hit?

5 ns

(b) What is the memory access time when you have a cache miss?

80 + 5 = 85 ns

(c) 10,000 accesses take 70 μs. What is the cache miss rate?

70 μs = 70,000 ns.
Average miss access time = 70,000/10,000 = 7 ns.
Average miss access time = miss_rate * 80 + 5
7 = miss_rate * 80 + 5
miss_rate = 2/80 = 2.5%

AY2016/17 Semester 2 Exam Q6

4-way set associative write-back cache.

Total 64KB. Each block = 8 words. Each word = 4 bytes. 32-bit addresses.

(d) How many bits per set do you require to store the tags?

32 bytes per block $\rightarrow$ byte offset = 5 bits.

Each set $\rightarrow$ 32 words = 128 bytes. #sets = 64KB/128 = 512
$\rightarrow$ set index = 9 bits.

Tag = 32 – 5 – 9 = 18 bits.

Therefore, #tag bits per set = 18×4 = 72 bits.

AY2016/17 Semester 2 Exam Q6

4-way set associative write-back cache.

Total 64KB. Each block = 8 words. Each word = 4 bytes. 32-bit addresses.

(e) What is the total amount of static RAM required to implement this cache?

Each block has 18 tag bits, 1 valid bit and 1 dirty bit → 20 bits.

In total, 512 (sets) $\times$ 4 (blocks/set) $\times$ 20 = 40,960 bits
= 5120 bytes = 5 KB.

Therefore, 64KB + 5KB = 69KB.

AY2016/17 Semester 2 Q4

(a) Using the instruction set given, write the Zephyr assembly language equivalent of this program. Ensure that your code is properly commented.

```
for (i=0; i<5; i++)
  if (x[i]<3)
    x[i] = x[i]+5;
```

```
       LDI $1, 0          ; Initialize index i to 0
       LDI $2, 5          ; Loop limit of 5
       LDI $3, x          ; Base address of array x
       LDI $4, 3          ; Our compare value
Label: LW $5, $3          ; Load array element
       BGT $5, $4, Out    ; Exit if bigger than 3
       BEQ $5, $4, Out    ; Exit if equal to 3
       PUSH $5            ; Add 5 to array elt
       PUSHI 5            ;
       ADD                ;
       POP $5             ; Get result
       SW $5, $3          ; Store it back to array
Out:   INCW $3            ; Increment base by 4
       INC $1             ; Increment index by 1
       BLT $1, $2, Label  ; Repeat if < 5
```

`LDI $y, c:` Load Immediate c into register y.

`LW $y, $x:` Load Word at address in $x into register y.

`SW $y, $x:` Store Word in register y into address in register x.

`BEQ/BGT/BLT $x, $y, disp:` Branch if register x equal/greater than/less than y.

`INC/DEC $y:` Increment/decrement register y by 1.

`INCW/DECW $y:` Increment/decrement register y by 4.

AY2016/17 Semester 2 Q4

(b) Sketch the instruction formats for all 6 classes, assuming that all 32 bits of a Zephyr instruction word are utilized fully, and that we maximize the number of opcode bits.

A: 32-bit operand

B: 29-bit operand, 3-bit register

C: 16-bit operand, 16-bit constant

D: 13-bit operand, 3-bit register, 16-bit constant

E: 26-bit operand, 3-bit register 1, 3-bit register 2

F: 10-bit operand, 3-bit register 1, 3-bit register 2, 16-bit displacement.

AY2016/17 Semester 2 Q4

(c) If we utilize an expanding opcode scheme for Zephyr, what is the maximum number of opcodes possible, assuming that there are at least one instruction in each class?

We want to maximize the number of Class A instructions but must leave one special opcode each for the remaining classes.

For these special opcodes, the remaining opcode bits cannot be used as they are part of "something else", e.g. register numbers.

So # of unusable bits for each opcode class is:

B: (32 – 29) = 3 bits
C: (32 – 16) = 16 bits
D: (32 – 13) = 19 bits
E: (32 – 26) = 6 bits
F: (32 – 10) = 22 bits

So, total number of opcodes is: $\mathbf{2^{32} - 2^{3} - 2^{16} - 2^{19} - 2^{6} - 2^{22} + 5}$

The +5 is the one instruction each in the remaining 5 classes.

AY2016/17 Semester 2 Q4

(d) What is the minimum number of opcodes possible, assuming that there are at least one instruction in each class?

We maximize the most restrictive case but leave opcodes for the remaining 5 classes.

In this case Class F is the most restrictive class since it has only 10 opcode bits. We reserve 5 opcodes for the remaining classes.

So # of opcodes = $2^{10} - 5 + 5 = 2^{10}$ = **1024 opcodes**.

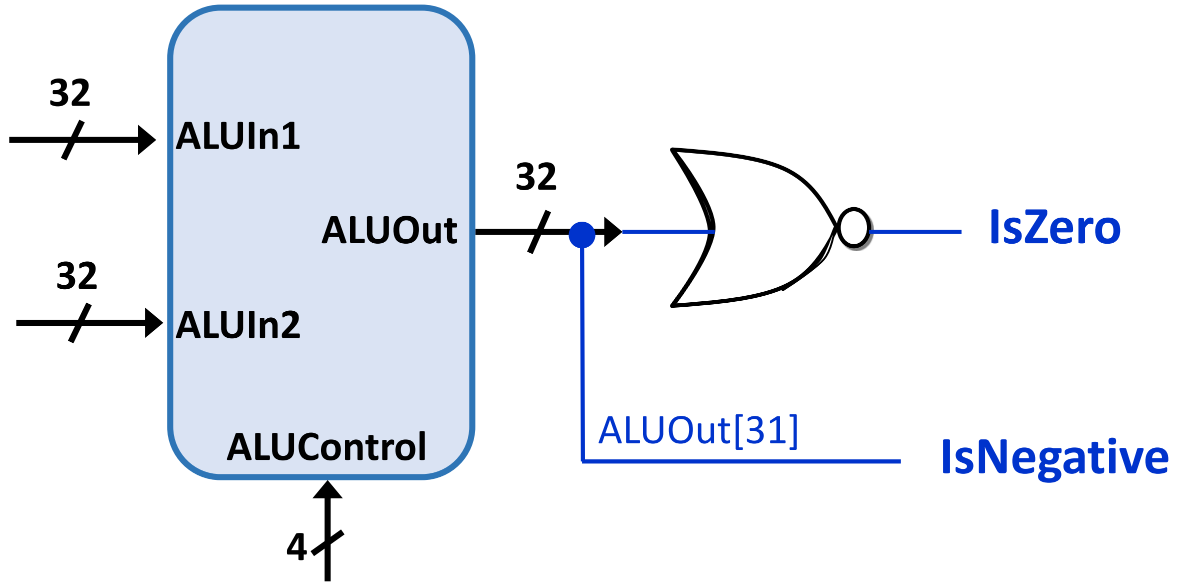
AY2016/17 Semester 2 Q5

To support 2 new instructions: BLT (Branch on Less Than) and BGT (Branch on Greater Than).

BLT:	0x08	rs	rt	displacement
BGT:	0x12	rs	rt	displacement

(a) The ALU is shown below. Show, by adding AT MOST ONE 32-input logic gate and any additional wires, how to generate the **IsZero** and **IsNegative** signals.

- IsZero signal is 1 when ALUIn1 – ALUIn2 = 0
- IsNegative signal is 1 when ALUIn1 – AluIn2 < 0.

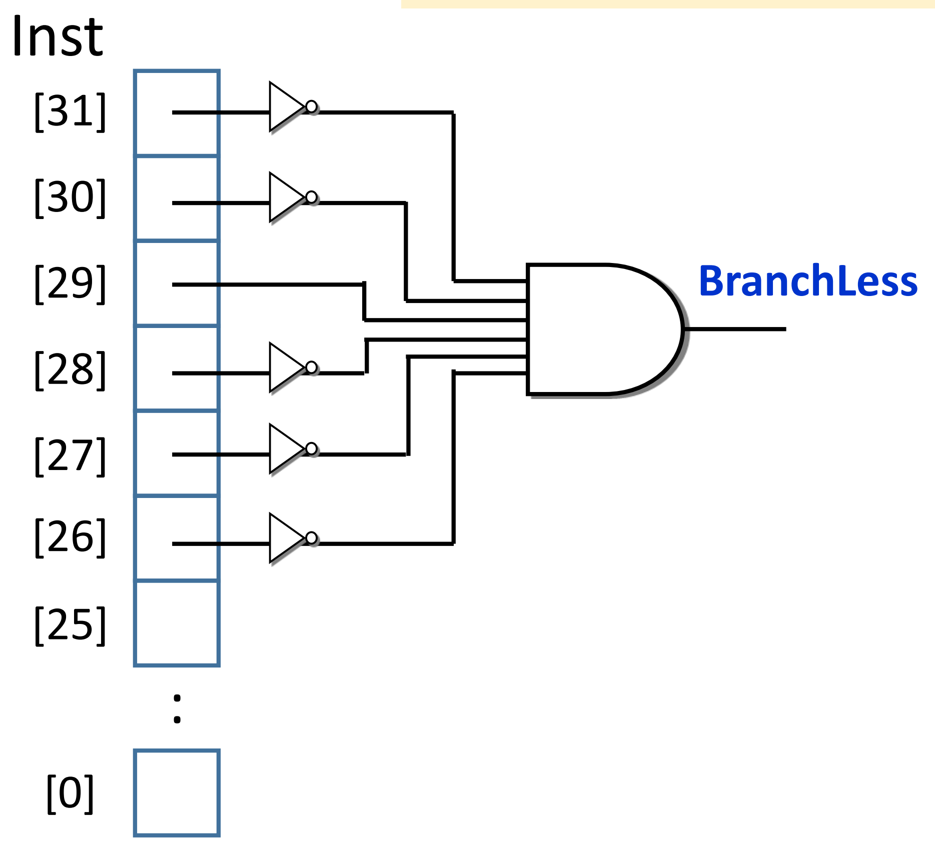


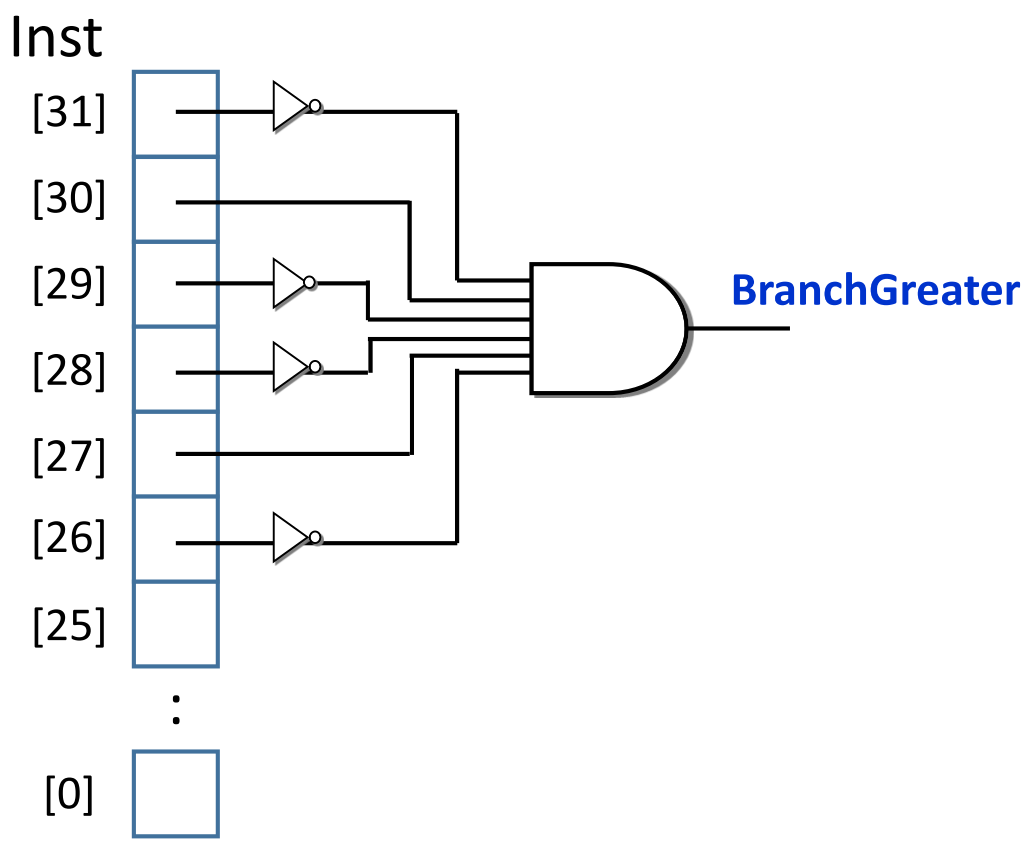
AY2016/17 Semester 2 Q5

(b) The CONTROL unit in the datapath must now generate **BranchLess** and **BranchGreater** signals from the instruction bits. Sketch the combinational circuits to generate these signals.

BLT: 0x08 = 00 1000

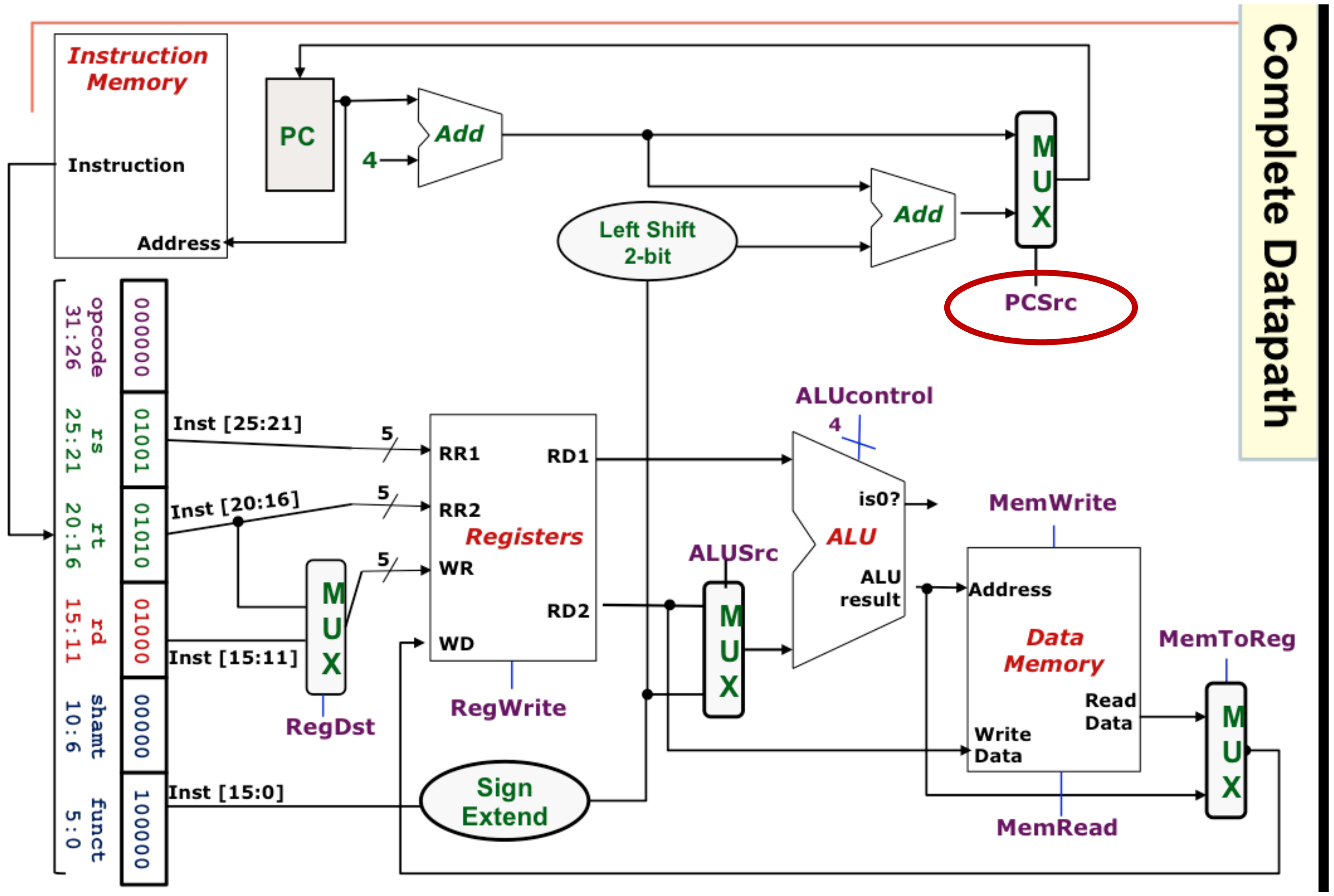
BGT: 0x12 = 01 0010

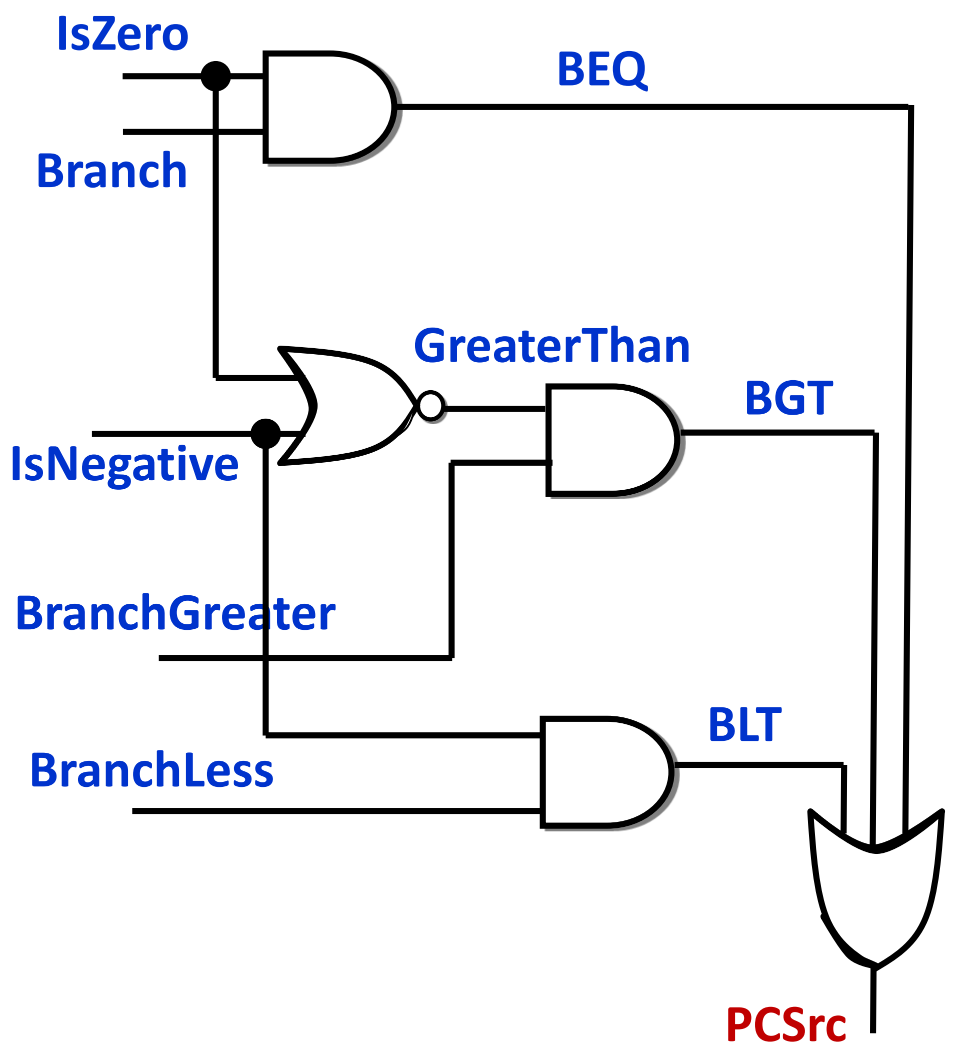




AY2016/17 Semester 2 Q5

(c) Sketch the combinational circuit needed to generate the **PCSrc** control signal to support the BEQ, BLT and BGT instructions.





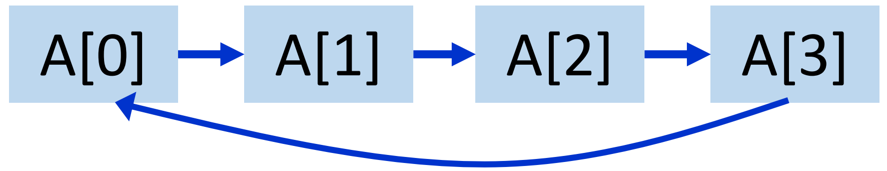
AY2015/16 Semester 2

AY2015/16 Semester 1 Q11

```
       addi $t1, $a0, 12    # I1
       lw   $t0, 12($a0)    # I2
Loop:  lw   $t2, -4($t1)    # I3
       sw   $t2, 0($t1)     # I4
       addi $t1, $t1, -4    # I5
       bne  $t1, $a0, Loop  # I6
       sw   $t0, 0($t1)     # I7
```

(a) $a0 stores the base addr of integer array A. What does the code do?

Rotates A[0], A[1], A[2], A[3] one element to the right.



(b) If first instruction executed is I1, which is the seventh instruction executed?

Instruction I3.

AY2015/16 Semester 1 Q11

```
      addi $t1, $a0, 12    # I1
      lw   $t0, 12($a0)    # I2
Loop: lw   $t2, -4($t1)    # I3
      sw   $t2, 0($t1)     # I4
      addi $t1, $t1, -4    # I5
      bne  $t1, $a0, Loop  # I6
      sw   $t0, 0($t1)     # I7
```

(c) No data forwarding.
Branch resolved at stage 4.

Total number of cycles:
$3+(3\times11)+5 = \mathbf{41}$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
I1	F	D	E	M	W																
I2		F	D	E	M	W															
I3			F	-	D	E	M	W													
I4				F	-	-	-	D	E	M	W										
I5					F	-	-	-	D	E	M	W									
I6						F	-	-	-	-	-	D	E	M	W						
I3															F	D	E	M	W		

AY2015/16 Semester 1 Q11

```
      addi $t1, $a0, 12   # I1
      lw   $t0, 12($a0)   # I2
Loop: lw   $t2, -4($t1)   # I3
      sw   $t2, 0($t1)    # I4
      addi $t1, $t1, -4   # I5
      bne  $t1, $a0, Loop # I6
      sw   $t0, 0($t1)    # I7
```

(d) With data forwarding.
Branch resolved at stage 2, predicted taken.

Total number of cycles = 23

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
I1	F	D	E	M	W																
I2		F	D	E	M	W															
I3			F	D	E	M	W														
I4				F	D	E	M	W													
I5					F	D	E	M	W												
I6						F	-	D	E	M	W										
I3							F	-	D	E	M	W									

AY2015/16 Semester 1 Q12

Direct-mapped cache, 8 blocks.
Each block 4 words.

(a) What is the cache hit rate of the code?

```
int sum = 0;
for (int i=0; i<32; i++) {
    sum = sum + (A[i] * B[i]) - C[i];
}
```

Addresses:
A: 0x00000080; *B*: 0xFFFF0040; *C*: 0x12345688

0x00000080 = … 0000 00001 000 0000
0xFFFF0040 = … 0000 00000 100 0000
0x12345688 = … 0101 01101 000 1000

First four iterations:
A[0]..A[3] mapped to block 0.
B[0]..B[3] mapped to block 4.
C[0], C[1] mapped to block 0,
C[2], C[3] mapped to block 1.

A[0] (miss), A[1] (miss), A[2] (miss), A[3] (hit).
B[0] (miss), B[1]..B[3] (hits)
C[0] (miss), C[1] (miss), C[2] (miss), C[3] (hit).

Hit rate = 5/12

AY2015/16 Semester 1 Q12

```
int sum = 0;
for (int i=0; i<32; i++) {
    sum = sum + (A[i] * B[i]) - C[i];
}
```

(b) Fill in the content of the cache after executing the code.

A:	… 0000 00001 000 0000
B:	… 0000 00000 100 0000
C:	… 0101 01101 000 1000

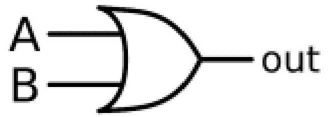
First four iterations:
A[0]..A[3] mapped to block 0.
B[0]..B[3] mapped to block 4.
C[0], C[1] mapped to block 0,
C[2], C[3] mapped to block 1.

Block 0	C[30]	C[31]	unknown	unknown
Block 1	B[20]	B[21]	B[22]	B[23]
Block 2	B[24]	B[25]	B[26]	B[27]
Block 3	B[28]	B[29]	B[30]	B[31]
Block 4	A[16]	A[17]	A[18]	A[19]
Block 5	A[20]	A[21]	A[22]	A[23]
Block 6	A[24]	A[25]	A[26]	A[27]
Block 7	A[28]	A[29]	A[30]	A[31]

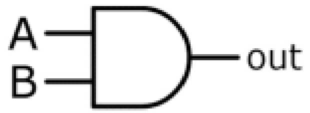
AY2021/22 Semester 1
CS2100 Exam

- Date: 20 November 2021, Saturday
- Time: 9 – 11am (set aside 8 – 1pm)
- Format
  - Q0: 3 marks
  - Part A: 6 MCQs (12 marks)
  - Part B: 5 MRQs (15 marks)
  - Part C: 5 long questions (70 marks)
- 3-page answer sheets provided (optional)
- Submit a single pdf file

OR gate

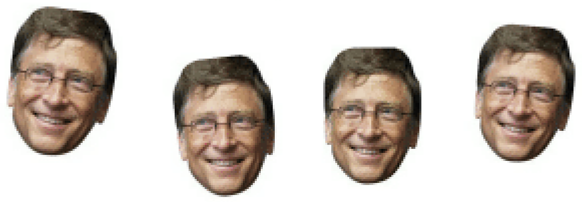


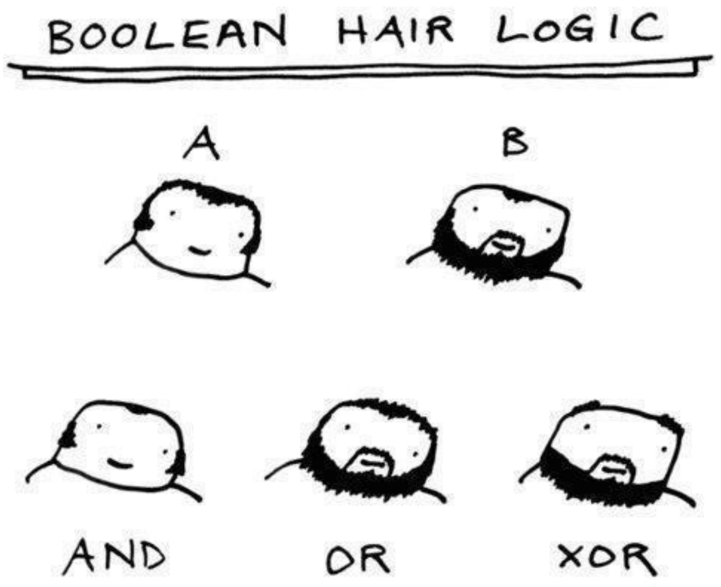
AND gate

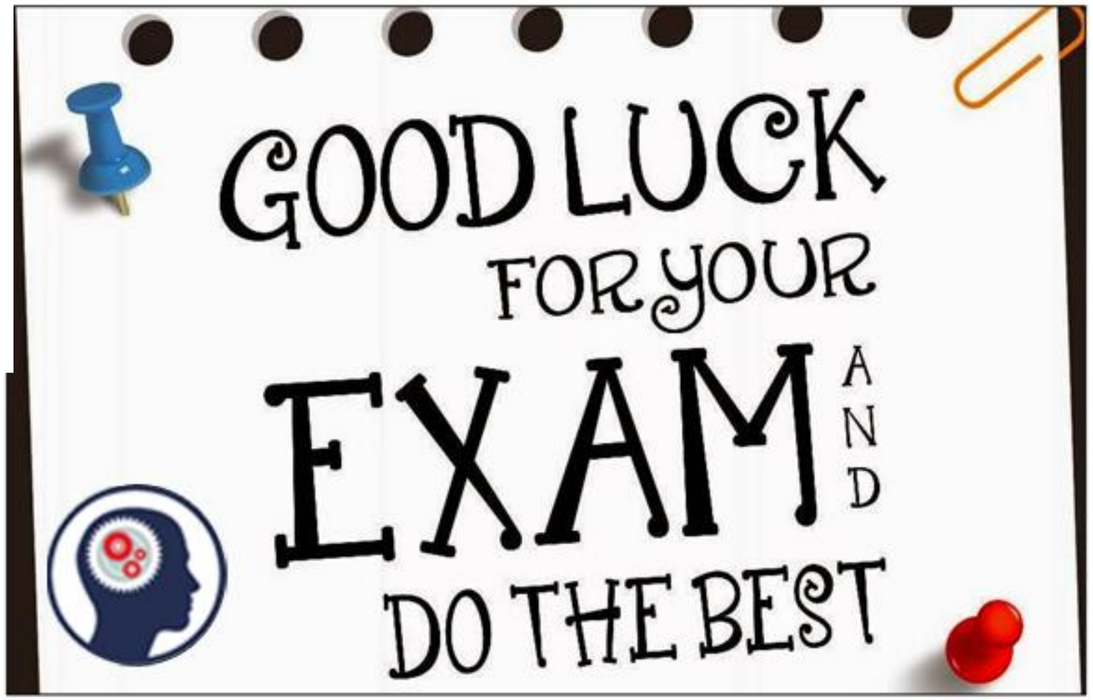


COL gate

BILL gates







END OF FILE