17/18

1. **[10 marks]**

(a) Write the output of the following C program. [4 marks]

```
#include <stdio.h>

typedef struct {
    int  val;
    char ch[2];
} rec_t;

void process1(rec_t *);
void process2(rec_t);

int main(void) {
    rec_t st[2] = {{11,{'A','B'}}, {22,{'C','D'}}};

    process1(&st[1]);
    process2(st[0]);
    printf("%d %c\n", st[0].val, st[0].ch[0]);
    printf("%d %c\n", st[1].val, st[1].ch[1]);
    return 0;
}

void process1(rec_t *para) {
    para->val = 33;
    para->ch[0] += ('a' - 'A') + 1;
    para->ch[1] += ('a' - 'A') + 2;
}

void process2(rec_t para) {
    para.val = 44;
    para.ch[0] += ('a' - 'A') + 3;
    para.ch[1] += ('a' - 'A') + 4;
}
```

st = [11 [A,B],
~~22~~ [C,D]
33 [d,e]]

Output: 11 A
33 e F

(b) Given the following hexadecimal representation in IEEE 754 single-precision floating-point number system:

42F64000

0100 0010 1111 0110 0100 0000 0000 0000
133

What is the decimal value it represents? [3 marks]

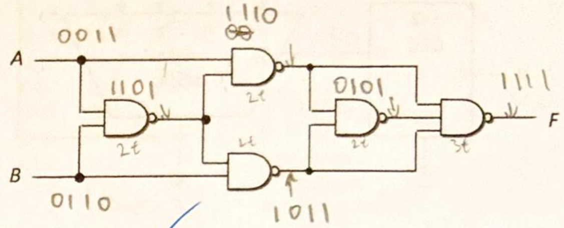
1.1110111001×2^6

1111011.001

123.125

1. *(continue...)*

(c) Given the logic circuit below:



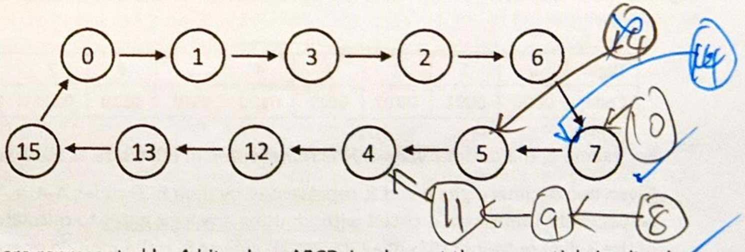
(i) What is *F*? F is 1. [2 marks]

(ii) What is the circuit propagation delay if the propagation delay of a NAND gate with fan-in of *n* is *nt*? [1 mark]

2+2+2+3 = 9t

2. **[15 marks]**

A sequential circuit goes through the following states, whose state values are shown in decimal:



The states are represented by 4-bit values *ABCD*. Implement the sequential circuit using a *D* flip-flop for *A*, a *D* flip-flop for *B*, a *T* flip-flop for *C*, and a *JK* flip-flop for *D*.

a. Write out the **simplified SOP expressions** for all the flip-flop inputs. [10 marks]

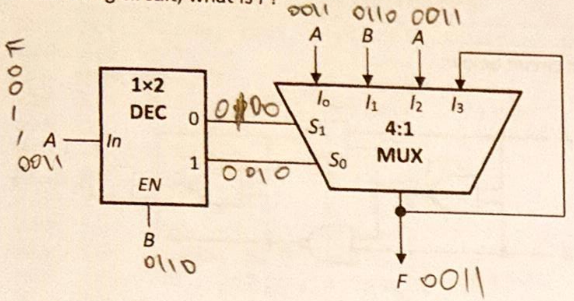
b. Implement your circuit according to your simplified SOP expressions obtained in part (a). Complete the given state diagram on the Answer Booklet, by indicating the next state for each of the five unused states. [5 marks]

Refer to other sheet.

3. **[20 marks]**

(a) Given the following circuit, what is *F*? [4 marks]

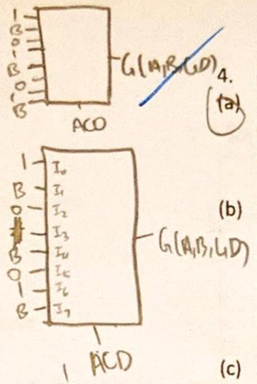
AB S_1S_0 $I_0I_1I_2I_3$ F
00 0 0 0 0 0 0
01 1 0 0 1 0 0
10 0 0 1 0 1 1
1~~1~~ 0 1 1 1 1 1



AB F
00 0
01 0
11 1
10 1

F = A.

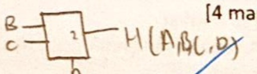
(b) Given $G(A,B,C,D) = \Pi M(1, 2, 6, 8, 9, 11, 13)$, implement *G* using a single 8:1 multiplexer without any additional logic gates. Complemented literals are not available. [4 marks]



0
0001
0010
0110
1000
1001
1011
1101

1
0000
0011
0100
0101
0111
1010
1100
1111

(c) Given $H(A,B,C,D) = \Sigma m(12, 13)$, implement *H* using a single 2×4 active high output decoder with 1-enable, without any additional logic gates. Complemented literals are not available. [4 marks]



(d) The BCD code (also known as 8421 code) values for the ten decimal digits are given below:

Digit:	0	1	2	3	4	5	6	7	8	9
Code:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

5×A 0000 0101 0001 0000

For example, the decimal value 396 is represented in BCD code as 0011 1001 0110.

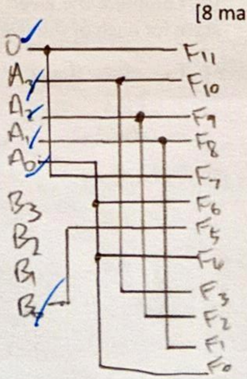
Given two decimal digits *A* and *B*, represented by their BCD codes $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ respectively, implement a circuit without using any logic gates to calculate the BCD code of the 3-digit output of (51×*A*) + (20×(*B*%2)), where % is the modulo operator. Name the outputs $F_{11}F_{10}F_9F_8$ $F_7F_6F_5F_4$ $F_3F_2F_1F_0$. You are free to use the logical constants 0 and 1.

0 A_3 A_2 A_1 0 A_0 B_0 A_0 A_3 A_2 A_1 A_0

For example, if *A*=2 (or 0010 in BCD) and *B*=7 (or 0111 in BCD), then (51×*A*) + (20×(*B*%2)) = 122 or 0001 0010 0010 in BCD. Hence, the circuit is to produce the output 0001 0010 0010 for the inputs 0010 and 0111.

(*Hint:* To help you, you may fill in the table on the Answer Booklet that computes 5×*A*. This table is worth 2 marks.)

[8 marks]



4. **[12 marks]**

1 i-format rest r-format $2^{12}-2^6 = 4032$

(a) Suppose MIPS instructions in R-format must use the following five opcodes (in decimal): 0, 1, 16, 17 and 32, what is the maximum total number of instructions that can be supported in MIPS? [2 marks]

~~$2^6-5(2^6)$~~ = $2^6-5+5(2^6) = 379$

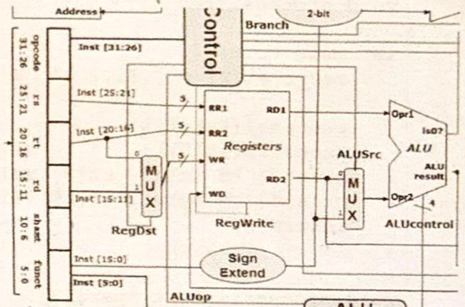
(b) Suppose due to a hardware defect in the datapath circuit, a stuck-at-0 fault occurs at **bit 6** of every MIPS instruction. This means that bit 6 of a MIPS instruction is always 0 regardless of what the instruction is originally. Devise a simple test using a MIPS instruction to discover this error. Explain your test. Keep your explanation clear and short, in no more than 2 sentences. [3 marks]

~~addi $t0, $t0, 5~~
~~addi $t0, $2, $0, 0~~.
~~j 0x0000000~~

addi $t0, $0, 64.
j 0x2000000
see if PC is at correct location

(c) The diagram on the right shows a portion of the datapth.

Suppose the stuck-at-0 fault occurs at the **ALUSrc** control signal. Assuming that **$t0** and **$t1** contains 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. [3 marks]



If no error correct, load value M[12] → $t1.
If error, load value M[12+34] into $t1.

Intended effect: $t1 → M[12]
Error effect: $t1 → $t0 [5] = 01000

~~addi $t1, $0, 12~~
addi $t0, $0, 5 } before
~~write data in to mem~~ write data t. m[12] = 1.
Check if $t0 value is 1 (correct) or 5 (wrong)

(d) The table below shows the ALUcontrol signal of the datapath we discussed in class.

Opcode	ALUop	Instruction operation	Funct field	ALU action	ALU control
lw	00	load word	xxxxxx	add	0010
sw	00	store word	xxxxxx	add	0010
beq	01	branch equal	xxxxxx	subtract	0110
R-type	10	add	100000	add	0010
R-type	10	subtract	100010	subtract	0110
R-type	10	AND	100100	AND	0000
R-type	10	OR	100101	OR	0001
R-type	10	set on less than	101010	set on less than	0111

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. [4 marks]

ALUop = 10.

bne results in branch taken if isZero = 0.

ALUop = 01 same as beq: Branch taken = !ALUop0 AND ALUop1 AND !isZero

5. **[15 marks]**

Study the MIPS program below. *A* and *B* are integer arrays whose base addresses are in **$s0** and **$s1** respectively. The arrays are of the same size *n* (number of elements). **$s2** contains the value *n*. The address of the first **beq** instruction is 0x0040003c.

```
# Q5.asm
.data
A:  .word 11, 9, 31, 2, 9, 1, 6, 10        +B +B +1  +1  +1 +1 +B +B
B:  .word 3, 7, 2, 12, 11, 41, 19, 35      [14,16,32,3,10,2,25,45]
n:  .word 8     011 010 1100 1101 1001 100011   [11,10,31,14,9,42,6,11]

.text
main: la    $s0, A      # $s0 is the base address of array A
      la    $s1, B      # $s1 is the base address of array B
      la    $t0, n      # $t0 is the addr of n (size of array)
      lw    $s2, 0($t0) # $s2 is the content of n
                                                     70
   3c beq   $s2, $zero, End     # Address:  0x0040003c
   40 addi  $t8, $s2, -1         0000 0000 0100 0000 0000 0000 0011 00
   44 sll   $t8, $t8, 2
Loop 48 add $t0, $s0, $t8   $t2 → A[i]
   4c add   $t1, $s1, $t8   $t3 → B[i]
   50 lw    $t2, 0($t0)
   54 lw    $t3, 0($t1)
   58 andi  $t4, $t3, 3     $t4 → 011 AND B[i]
   5c addi  $t4, $t4, -3    If 011 AND B[i] = 3, branch to A1
   60 beq   $t4, $zero, A1
   64 add   $t2, $t2, $t3   A[i] += B[i]
   68 j     A2
A1: 6c addi $t2, $t2, 1        A[i] += 1
A2: 70 sw   $t2, 0($t0)     i -= 2
      addi  $t8, $t8, -8
      slt   $t7, $t8, $zero
      beq   $t7, $zero, Loop
End:  li    $v0, 10             # system call code for exit
      syscall
```

```
int i = n-1; ✓
do{ for(... )
  if ( B[i] & 3 -3 == 0){
     A[i] = A[i] + B[i];
  } else{
     A[i] += 1;
  }
  i-=2;
} while( i >= 0 );
```

A 1010
B 1011
C 1100
D 1101
E 1110
F 1111

a. Fill in the missing instruction (the fourth line in the program text) to store the value of *n* into **$s2**. Do not use any pseudo-instruction. [1 mark]

b. Fill in the values of array *A* after the execution of the code. [4 marks]

c. Write an equivalent C code that does the same work. Use variables *A* and *B* for the arrays, and *n* for the size of the array. You do not need to declare *A*, *B* and *n*. [4 marks]

Give the instruction encoding in hexadecimal for the following 3 instructions:

d. **sll $t8, $t8, 2** (Note: rs = 0) 000000 11000 00000 11000 00010 000000 0x0300B080 [2 marks]

e. **j A2** 000010 0000 0100 0000 0000 0000 0111 00 0x0810001C [2 marks]

f. **slt $t7, $t8, $zero** 000000 11000 00000 10001 000 00101010 0x0300882A [2 marks]

sll $t8,$t8, 2 op 000000 rs=00000 rt= 11000 rd= 11000 shamt=00010 funct= 000000
0x 00180080

slt $t7 $t8 $0 op 000000 rs= 11000 rt= 00000 rd=01111 shamt=0 funct= 101010
0x0300782A

6. **[14 marks]**

Refer to the same MIPS code in the previous question, except that now we focus only on a section of the code which is reproduced below:

```
      beq   $s2, $zero, End     # Inst1
      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
LW    andi  $t4, $t3, 3         # Inst8
      addi  $t4, $t4, -3        # Inst9
RAW   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
RAW   slt   $t7, $t8, $zero     # Inst16
      beq   $t7, $zero, Loop    # Inst17
End:
```

IF ID Ex MEM WB

Assuming a 5-stage MIPS pipeline system with forwarding and early branching, that is, the branch decision is made at the ID stage. No branch prediction is made and no delayed branching is used. For the jump (j) instruction, the computation of the target address to jump to is done at the ID stage as well.

Assume also that the first **beq** instruction begins at cycle 1.

a. Suppose arrays *A* and *B* now each contains 200 positive integers. What is the minimum number and maximum number of instructions executed? (Consider only the above code segment from Inst1 to Inst17.) 13×200+3 = 2603 [2 marks]

min 12×100+3=1203 max = 13×100+3 = 1303.

b. List out the instructions where some stall cycle(s) are inserted in executing that instruction in the pipeline. These include delay caused by data dependency and control hazard. You may write the instruction number InstX instead of writing out the instruction in full. Inst 8 Inst 1, 10, 17, 12. [6 marks]

control: ~~1, 10, 12, 17~~: 2, 11, 13, ~~12~~ 4, 14. data: 8, 10, 17.

c. How many cycles does one iteration of the loop (from Inst1 to Inst17) take if the **beq** instruction at Inst10 branches to *A1*? You have to count until the WB stage of Inst17. [3 marks]

22 cycles. 24

d. How many cycles does one iteration of the loop (from Inst1 to Inst17) take if the **beq** instruction at Inst10 does not branch to *A1*? You have to count until the WB stage of Inst17. [3 marks]

24 cycles 26

7. **[14 marks]**
Refer to the same MIPS code in the previous two questions:

```
                                                              0011 1100
      beq  $s2, $zero, End    # Inst1, Address: 0x0040003c
      addi $t8, $s2, -1       # Inst2                    40
      sll  $t8, $t8, 2        # Inst3                    44
Loop: add  $t0, $s0, $t8      # Inst4                    48   01001000
      add  $t1, $s1, $t8      # Inst5                    4C
      lw   $t2, 0($t0)        # Inst6                    50   0101 0000
      lw   $t3, 0($t1)        # Inst7                    54
      andi $t4, $t3, 3        # Inst8                    58
      addi $t4, $t4, -3       # Inst9                    5C   1024x
      beq  $t4, $zero, A1     # Inst10                   60
      add  $t2, $t2, $t3      # Inst11
      j    A2                 # Inst12
A1:   addi $t2, $t2, 1        # Inst13                   6C   0110 1100
A2:   sw   $t2, 0($t0)        # Inst14                   70
      addi $t8, $t8, -8       # Inst15                   74
      slt  $t7, $t8, $zero    # Inst16                   78
      beq  $t7, $zero, Loop   # Inst17                   7C
End:
```

A[0]
0x10001000
A[1023]
1FFC
0111 1111 1111 1100

B[0]
0x1003F100
B[1023]
00FC
01111 1100

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 (that is, when **lw** and **sw** instructions are executed).

a. How many bits are there in the index field? In the byte offset field? [2 marks]

5 ✓ 4 ✓

b. Which index is *A*[1023] mapped to? Which index is *B*[1023] mapped to? [4 marks]

~~16~~ 11100 = 28 31 ~~16~~ 4 15.

10001000
3FF0X
1004F00
0001 1111 0000 0000
1003F100
3FF00
10081100
0001 0001 0000 000

c. How many memory accesses in total are made for array *A*? For array *B*? [2 marks]

A: 1 read, 1 write. B only ~~reads~~ ~~256~~ 1024 ✓ ~~256~~ ~~512~~ 2048 512

d. What is the cache hit rate for array *A*? For array *B*? [2 marks]

75% ✓ 87.5% 50%. 256 miss 256 hits.

Total instructions = 3+ ~~12~~(512) = 6147.

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, that is, Inst1 through Inst17. [4 marks]

8 blocks index = 3 bit
offset = 3 bit

cold miss ↓ 1(14)

hit 1023(11)

~~ End OF PAPER ~~

cold. I1, 2, 4, 6, 8, 10, 14, 16, 17
~~I1 I~~

	W0	W1
000	conflict ~~I2~~	I3
001	I4	I5
~~0100~~	I6	I7
011	I8	I9
100	I10	
101		I13
110	~~I14~~	I15
111	I16	~~I1~~ I17

Hits: 11253
misses: 14

9 misses. I1, I17 conflict! ~~misses 7~~.

6147 - 9 = 6138 hits.

Forwarding Early branching at ID.

IF ID EX MEM WB

(This page is for your rough work.)

lw

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
I1 beq	IF	ID	EX	MEM	WB																									
I2 addi			IF	ID	EX	MEM	WB ✓																							
I3 sll				IF	ID	EX	MEM	WB																						
I4 add					IF	ID	EX	MEM	WB																					
I5 add						IF	ID	EX	MEM	WB																				
I6 lw							IF	ID	EX	MEM	WB																			
I7 lw								IF	ID	EX	MEM	WB																		
I8 andi									IF	ID		EX	MEM	WB ✓																
I9 addi										IF	ID	~~ID~~	EX	MEM	WB															
I10 beq A1											IF.	IF	ID ID	EX	MEM E	WB M	W +2 delay.													
I11 add														No Branch. IF	ID	EX	MEM	WB												
I12 J A2															IF	ID	EX	MEM	WB											
I13 A1: addi														Branch IF	ID	EX	MEM	WB												
I14 A2: sw															IF	ID	EX IF	MEM ID	WB EX	MEM	WB									
I15 addi																IF	ID	EX	MEM	WB										
I16 slt																	IF	ID	EX	MEM	WB									
I17 beq																		IF	ID	EX	MEM	WB			IF no branch.					

early branching: ID needs EX of I9.