

1(a) Let X be the base-16 number 1A. Write the number $-X$ in base-8 7's complement form. Your answer should have 4 base-8 digits, ie. use the offset origin which is displaced from the true origin by $10^4_{(8)}$.

1(b) $Z=Y/X$ where $X=14$, $Y=294577$ and all 3 numbers are base-16. Find Z rounded to the nearest integer.

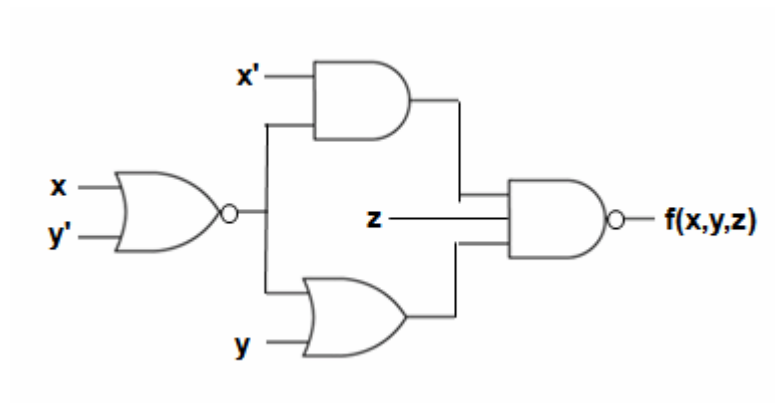
(a) $X=1A_{(16)}=32_{(8)}$. The 7's complement of 0032 is 7745.

(b)

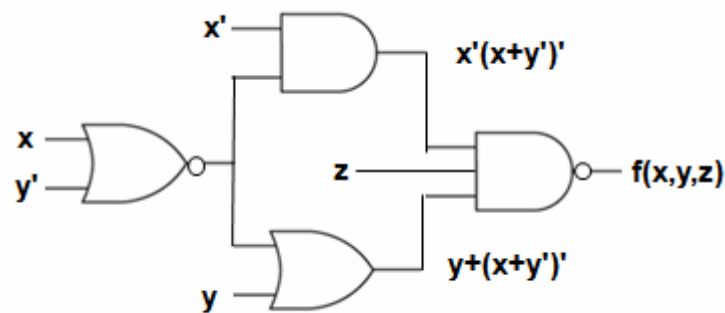
$$\begin{array}{r}
 21045 \text{ R } 13 \\
 14 \overline{) 294577} \\
 \underline{28} \\
 14 \\
 \underline{14} \\
 057 \\
 \underline{50} \\
 77 \\
 \underline{64} \\
 13
 \end{array}$$

1(a) <div style="text-align: center;">7745</div>	1(b) <div style="text-align: center;">21046</div>
---	--

2.



2(a) Express $f(x,y,z)$ as a DNF Boolean function.



$$\begin{aligned}
 \text{So } f &= (zx'(x+y)')(y+(x+y)')' \\
 &= z'+x+(x+y)'+(y+(x+y)')' \\
 &= z'+x+y'+y'(x+y) \\
 &= z'+x+y'
 \end{aligned}$$

2(b) Express $f(x,y,z)$ in minterm canonical form.

$$\begin{aligned}
 f &= (x+x')(y+y')z'+x(y+y')(z+z')+(x+x')y'(z+z') \\
 &= xyz'+xy'z'+x'yz'+x'y'z'+xyz+xy'z+x'y'z
 \end{aligned}$$

<p>2(a)</p> $f = x + y' + z'$	<p>2(b)</p> $f = xyz' + xy'z' + x'yz' + x'y'z' + xyz + xy'z + x'y'z$
-------------------------------	--

3.

$$f(w, x, y, z) = x + (x'yz' + z)' + (w + y + z)'$$

(a) Draw any circuit for $f(w, x, y, z)$ which uses only OR and AND gates. No bubbles or inverting amplifiers are allowed. Assume double-rail logic.

(b) Convert your circuit from OR and AND gates to one using just NAND gates.

(a) $f = x + (x'yz' + z)' + (w + y + z)'$

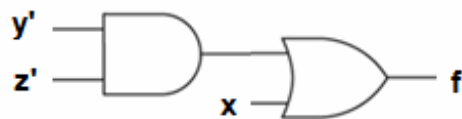
$$= x + (x'yz')'z' + w'y'z'$$

$$= x + (x + y' + z)z' + w'y'z' \quad \text{Can use this but easier to simplify first.}$$

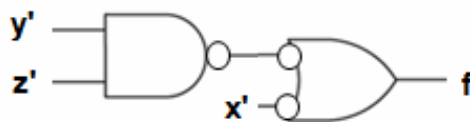
$$= x + y'z'$$

(b) Put bubbles at outputs of AND gates (making them type 1 NANDs), input of ORs (making them type 2 NANDs), then compensate.

3 (a)



3 (b)



4.

$$f(w, x, y, z) = \sum m(0, 1, 2, 8, 9, 10)$$

(a) Sketch the Karnaugh Map for f . Show the maximal subcubes.

(b) Find the minimal DNF realization for f .

(c) Find the minimal CNF realization for f .

(a,b) 2 max 1-cell subcubes: 4-corners ($x'z'$), rows 1&4 cols 1&2 ($x'y'$). Both are irredundant, so only one minimal DNF.

(c) 2 max 0-subcubes: full rows 2&3, full col 3. Both are irredundant, so only one minimal CNF.

4(a)

		yz			
		00	01	11	10
wx	00	1	1	0	1
	01	0	0	0	0
	11	0	0	0	0
	10	1	1	0	1

4(b) $x'z' + x'y'$

4(c) $x'(y'+z')$

5. Answer each part of this question. The parts are not related.

(a) Suppose $f(x,y,z)$ has a prime implicate $g(x,y,z)$. Does f imply g , or vice-versa? Illustrate your answer by giving any specific $f(x,y,z)$, identifying one of its prime implicates, and showing that your answer works for this example case.

f implies g . Consider for instance $f=(x+y+z)(x'+y'+z')$. Then $g=x+y+z$ is a prime implicate of f . If f is true, it must be the case that g is true, since $f=g*\text{something}$ and $g=0$ would imply $f=0$.

(b) Suppose each non-empty column of the Quine-McCluskey algorithm for a given f ends up with exactly one entry without a check mark (one unmatched entry). What does that tell us about its prime implicants?

It tells us that f has one prime implicant with n literals, one with $n-1$ literals etc. down to $n-m+1$, where n is the number of variables in f and m is the number of non-empty columns in the QM algorithm.

(c) Is every 1-cell in a Karnaugh Map necessarily covered by at least one essential prime implicant? If yes explain, if no give an example.

No, some 1-cells may be covered only by non-essential prime implicants. For instance, the cell in the KM below with the * in it is such a 1-cell.

			1
		1*	1
	1	1	

6. Design a circuit which produces the repeating output sequence shown in the table to the left. Starting from $Q_0Q_1=01$, after the next positive clock edge the outputs go to 10, then 00, then back to 01, and so on. Use a D flip-flop for Q_0 and T flip-flop for Q_1 (both positive edge-triggered) as shown in the answer box. The function tables for both types of flip-flop are shown to the right. Complete the circuit diagram begun in the answer box.

Flip-flop function tables

Q_0	Q_1
0	1
1	0
0	0

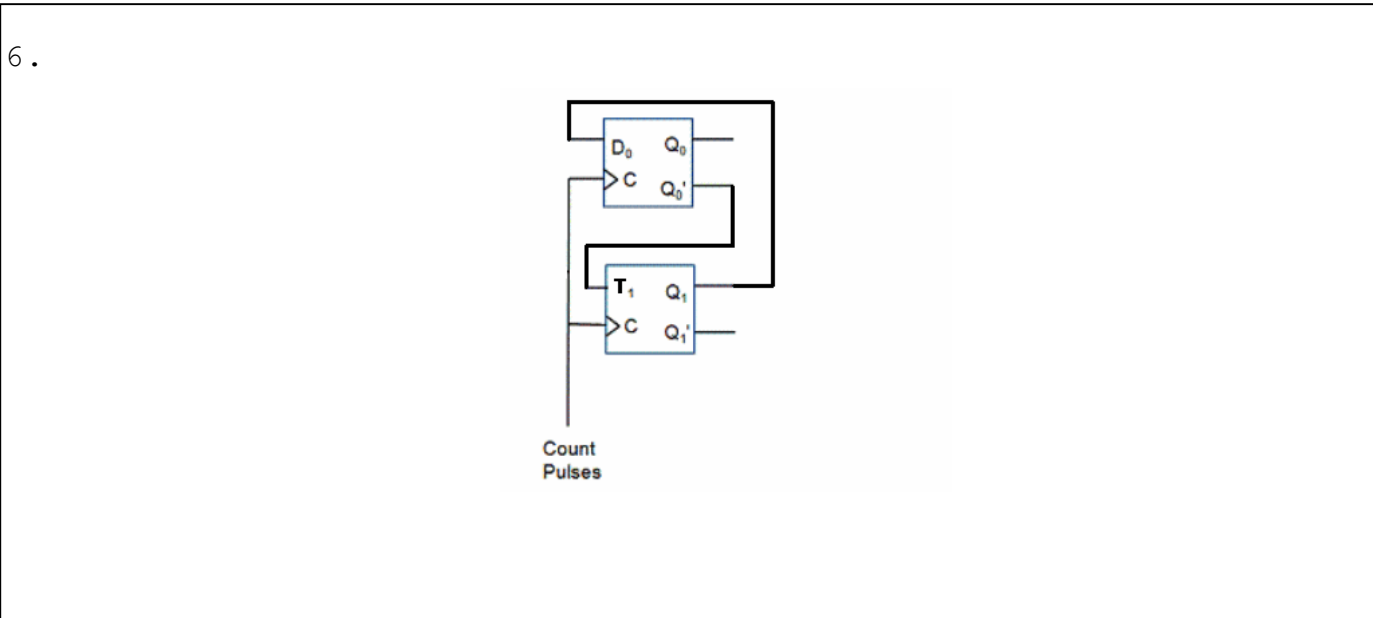
Inputs		Outputs	
D	C	Q^+	Q'^+
0	↑	0	1
1	↑	1	0
x	0	Q	Q'

Inputs		Outputs	
T	C	Q^+	Q'^+
0	↑	Q	Q'
1	↑	Q'	Q
x	0	Q	Q'

Excitation table:

Q_0	Q_1	D_0	T_1	Q_0^+	Q_1^+
0	0	0	1	0	1
0	1	1	1	1	0
1	0	0	0	0	0
1	1	-	-	-	-

So comparing the 2nd and 3rd cols, we want $D_0=Q_1$. And comparing 1st and 4th columns, we want $T_1=Q_0'$. We wire it that way in the solution box.



7. Write a MIPS assembler program that will read an integer from the console and store it at the label x. If it is ≤ 64 , the square of this input should be stored 6 words past x and the program should terminate. If it is > 64 , the message "warning:>64" should be printed on the console and the program should terminate.

DRAFT

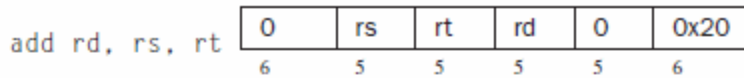
FINAL

	.data
	warning_msg:
	.asciiz "warning:input>64"
	x:.word 0
	.globl main
	.text
	main:
	li \$t0, 64
	li \$v0, 5
	syscall #get integer
	sw \$v0, x #store at x
	sub \$t0, \$v0, \$t0
	blez \$t0, leq64
	ori \$0, 0 #delay slot
	li \$v0, 4
	la \$a0, warning_msg
	syscall #print warning
	li \$v0, 10
	syscall #exit
	leq64:
	mul \$t1, \$v0, \$v0
	la \$t2, x
	sw \$t1, 24(\$t2) # store sq
	li \$v0, 10
	syscall #exit

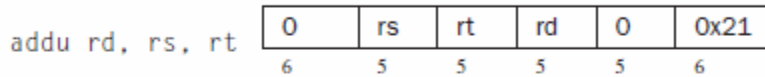
When you are done, cross out one of the columns and the code in the other column will be graded.

Reference Sheet

Addition (with overflow)

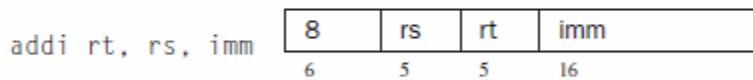


Addition (without overflow)

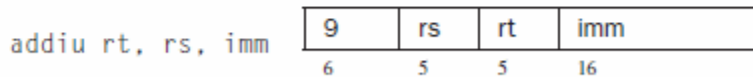


Put the sum of registers *rs* and *rt* into register *rd*.

Addition immediate (with overflow)

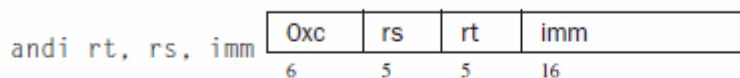


Addition immediate (without overflow)



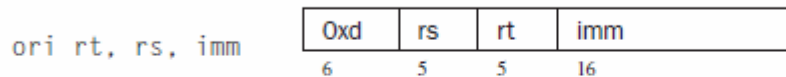
Put the sum of register *rs* and the sign-extended immediate into register *rt*.

AND immediate



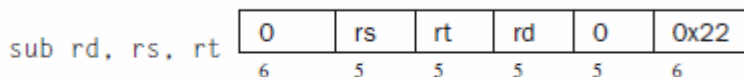
Put the logical AND of register *rs* and the zero-extended immediate into register *rt*.

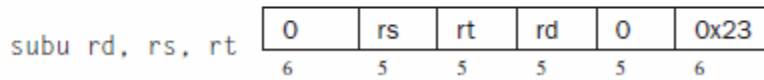
OR immediate



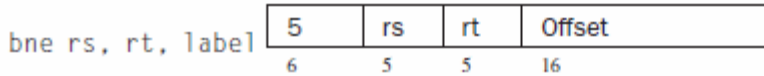
Put the logical OR of register *rs* and the zero-extended immediate into register *rt*.

Subtract (with overflow)



Subtract (without overflow)

Put the difference of registers *rs* and *rt* into register *rd*.

Branch on not equal

Conditionally branch the number of instructions specified by the offset if register *rs* is not equal to *rt*.

Branch on equal zero

`beqz rsrc, label` *pseudoinstruction*

Conditionally branch to the instruction at the label if *rsrc* equals 0.

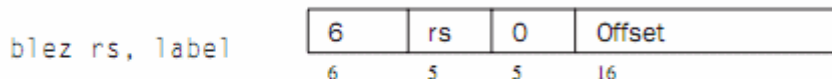
Branch on greater than equal

`bge rsrc1, rsrc2, label` *pseudoinstruction*

Branch on greater than equal unsigned

`bgeu rsrc1, rsrc2, label` *pseudoinstruction*

Conditionally branch to the instruction at the label if register *rsrc1* is greater than or equal to *rsrc2*.

Branch on less than equal zero

Conditionally branch the number of instructions specified by the offset if register *rs* is less than or equal to 0.

Jump

`j target`

2	target
6	26

Unconditionally jump to the instruction at *target*.

Load immediate

`li rdest, imm` *pseudoinstruction*

Move the immediate *imm* into register *rdest*.

Load word

`lw rt, address`

0x23	rs	rt	Offset
6	5	5	16

Load the 32-bit quantity (word) at *address* into register *rt*.

Load address

`la rdest, address` *pseudoinstruction*

Load computed *address*—not the contents of the location—into register *rdest*.

Load halfword

`lh rt, address`

0x21	rs	rt	Offset
6	5	5	16

Load unsigned halfword

`lhu rt, address`

0x25	rs	rt	Offset
6	5	5	16

Load the 16-bit quantity (halfword) at *address* into register *rt*. The halfword is sign-extended by `lh`, but not by `lhu`.

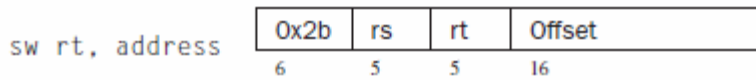
Multiply (without overflow)

`mul rd, rs, rt`

0x1c	rs	rt	rd	0	2
6	5	5	5	5	6

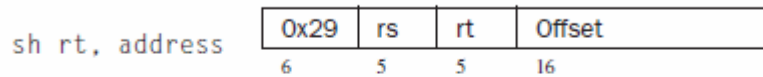
Put the low-order 32 bits of the product of *rs* and *rt* into register *rd*.

Store word



Store the word from register *rt* at *address*.

Store halfword



Store the low halfword from register *rt* at *address*.

Syscalls

Service	System call code	Arguments	Result
print_int	1	\$a0 = integer	
print_float	2	\$f12 = float	
print_double	3	\$f12 = double	
print_string	4	\$a0 = string	
read_int	5		integer (in \$v0)
read_float	6		float (in \$f0)
read_double	7		double (in \$f0)
read_string	8	\$a0 = buffer, \$a1 = length	
sbrk	9	\$a0 = amount	address (in \$v0)
exit	10		
print_char	11	\$a0 = char	
read_char	12		char (in \$a0)
open	13	\$a0 = filename (string), \$a1 = flags, \$a2 = mode	file descriptor (in \$a0)
read	14	\$a0 = file descriptor, \$a1 = buffer, \$a2 = length	num chars read (in \$a0)
write	15	\$a0 = file descriptor, \$a1 = buffer, \$a2 = length	num chars written (in \$a0)
close	16	\$a0 = file descriptor	
exit2	17	\$a0 = result	

FIGURE A.9.1 System services.