

**CSC205**  
**Project 1: Assembly Language Programming in MARIE**  
**Topic: Parity**

**Total possible points: 20 (+5 bonuses)**

**A. Introduction**

The objective in this project is to implement a parity checker in MARIE.

Parity checking is very useful in applications of network communication and data storage. RAM memory typically incorporates a parity checker on board. RAID uses parity as one method of safeguarding data integrity.

Let X be an input string. An even parity checker outputs the value 1 if X contains an odd number of 1's in the input X. The checker outputs the value 0 otherwise.

Before the invention of Unicode, ASCII was the standard representation of characters. In the ASCII system, there are in total 128 characters (including digits, letters in uppercase and lowercase, punctuations and selected special characters). Table 1 shows the printable ASCII characters (excluding whitespace) that are recognized by MARIE. Each character has 7 bits. When the characters are stored on a disk or transmitted across a network, a parity bit may be added to the end of each code to make up for 8 bits.

Glyph	Dec	Binary	Glyph	Dec	Binary	Glyph	Dec	Binary
			@	64	100 0000	`	96	110 0000
!	33	010 0001	A	65	100 0001	a	97	110 0001
"	34	010 0010	B	66	100 0010	b	98	110 0010
#	35	010 0011	C	67	100 0011	c	99	110 0011
\$	36	010 0100	D	68	100 0100	d	100	110 0100
%	37	010 0101	E	69	100 0101	e	101	110 0101
&	38	010 0110	F	70	100 0110	f	102	110 0110
'	39	010 0111	G	71	100 0111	g	103	110 0111
(	40	010 1000	H	72	100 1000	h	104	110 1000
)	41	010 1001	I	73	100 1001	i	105	110 1001
*	42	010 1010	J	74	100 1010	j	106	110 1010
+	43	010 1011	K	75	100 1011	k	107	110 1011
,	44	010 1100	L	76	100 1100	l	108	110 1100
-	45	010 1101	M	77	100 1101	m	109	110 1101
.	46	010 1110	N	78	100 1110	n	110	110 1110
/	47	010 1111	O	79	100 1111	o	111	110 1111
0	48	011 0000	P	80	101 0000	p	112	111 0000
1	49	011 0001	Q	81	101 0001	q	113	111 0001
2	50	011 0010	R	82	101 0010	r	114	111 0010
3	51	011 0011	S	83	101 0011	s	115	111 0011
4	52	011 0100	T	84	101 0100	t	116	111 0100
5	53	011 0101	U	85	101 0101	u	117	111 0101
6	54	011 0110	V	86	101 0110	v	118	111 0110
7	55	011 0111	W	87	101 0111	w	119	111 0111
8	56	011 1000	X	88	101 1000	x	120	111 1000

9	57	011 1001	y	89	101 1001	y	121	111 1001
:	58	011 1010	z	90	101 1010	z	122	111 1010
;	59	011 1011	[	91	101 1011	{	123	111 1011
<	60	011 1100	\	92	101 1100		124	111 1100
=	61	011 1101	]	93	101 1101	}	125	111 1101
>	62	011 1110	^	94	101 1110	~	126	111 1110
?	63	011 1111	_	95	101 1111			

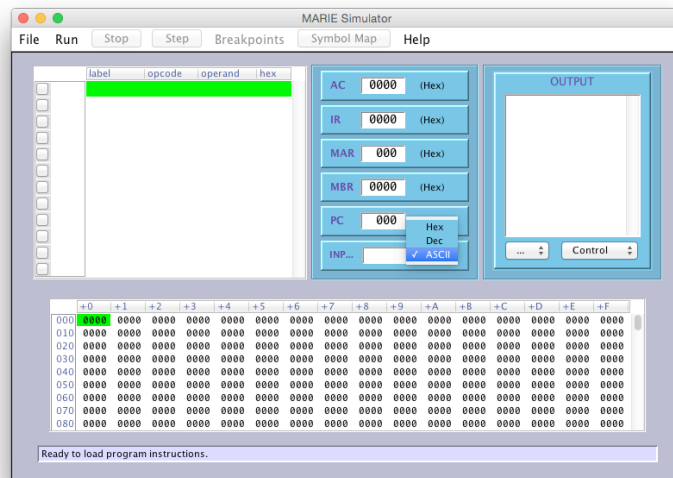
**Table 1: Printable ASCII characters**

## B. Problem description

Write a parity checker for the ASCII system. The parity checker should repeatedly execute a loop that performs the following tasks:

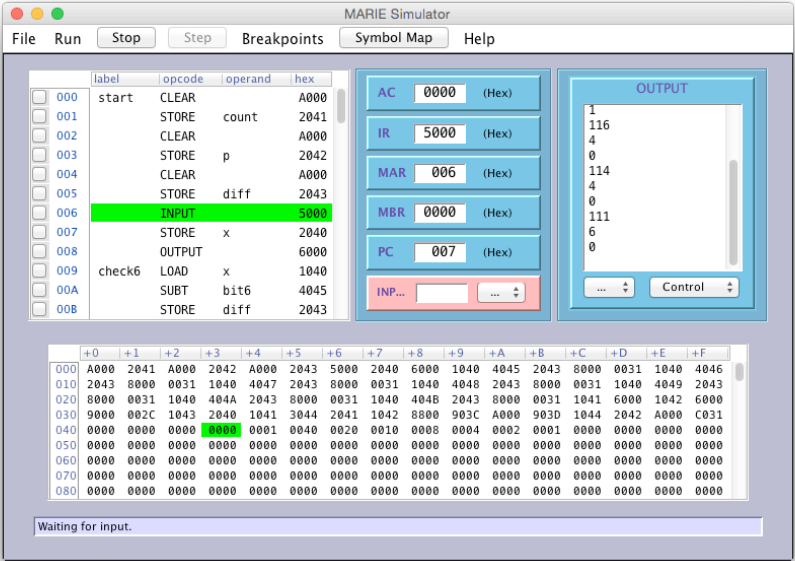
1. Ask the user for an input X, which can be any printable ASCII character from Table 1;
2. Output the decimal code of X;
3. Output the total number of 1's that appears in the binary code of X;
4. Output the parity bit which, when added to the binary code of X, will make the number of 1's even.

The user may enter input directly as ASCII characters by setting the input mode on the simulator, as shown in the picture below.

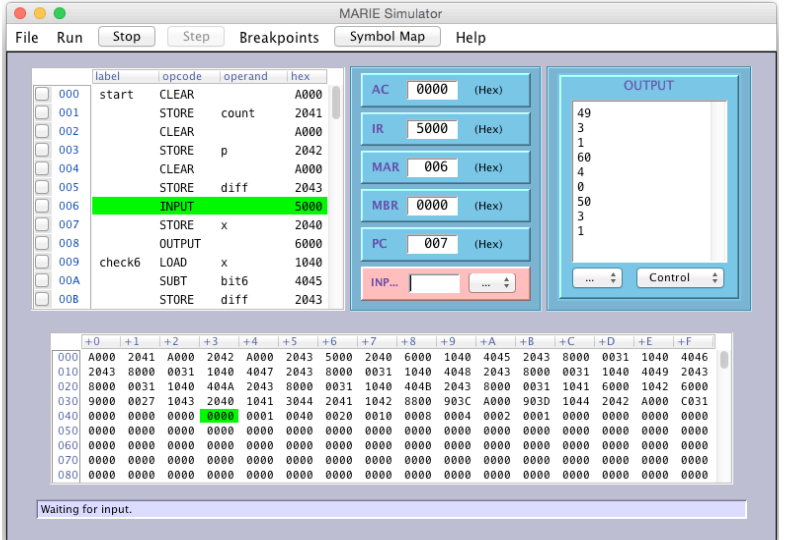


## C. Examples

Example 1:

Input sequence	Program's output sequence:	A screenshot showing the status of the program at the end of the input sequence:
I n t r o	73 3 1 110 5 1 116 4 0 114 4 0 111 6 0	

Example 2:

Input sequence	Program's output sequence:	A screenshot showing the status of the program at the end of the input sequence:
1 < 2	49 3 1 60 4 0 50 3 1	

#### **D. Automated IO testing**

Automated IO testing enables your program to be tested with a vast number of inputs, thus ensuring your program behavior meets the designated requirements such as correctness of response and efficiency.

You may test your program using an IO test package for MARIE at <https://github.com/huiannie/MarieTest>

#### **E. Deliverables**

Please test all the algorithms you write to make sure that they work as desired. Then submit the following items through Blackboard:

1. An electronic copy of your program source code, with proper documentations.
2. A screenshot of your program's execution result for the following input sequence:  
    Z !
3. Results of automated IO testing.

#### **F. Grading**

1. [3 pts] Correctness of your count of 1's for each input.
2. [3 pts] Correctness of your parity bit for each input.
3. [3 pts] Correctness of your Input of X and the display of X in decimal.
4. [2 pts] Consistence of your program's execution results shown in screenshot.
5. [2 pts] Evidence of conducting automated IO test.
6. [2 pts] Efficiency (your program must produce results within the time limit set by the automated IO test driver).
7. [5 pts] Documentation of your source code.
8. [5 bonus pts] The reverse engineered source code of the sample program found in Appendix A.

## Appendix A (Source: parity-nested.mas)

Convert machine code to assembly code. Where an assembly code requires an argument, record the address of the argument in the symbol table.

Pass (1)

Symbol table Page 1

Address	Label	Machine code	Assembly code (best guess)	Hash to	Address	Symbol
000		003F	JNS 03F	S1	03F	S1
001		104A	LOAD 04A	S2	04A	S2
002		404F	SUBT 04F	S3	04F	S3
003		204D	STORE 04D	S4	04D	S4
004		8000	SKIPCOND 000			S5
005		0029				S6
006		104A				S7
007		4050				S8
008		204D				S9
009		8000				S10
00A		0029				S11
00B		104A				S12
00C		4051				S13
00D		204D				S14
00E		8000				S15
00F		0029				S16
010		104A				S17
011		4052				S18
012		204D				S19
013		8000				
014		0029				
015		104A				
016		4053				
017		204D				
018		8000				
019		0029				
01A		104A				
01B		4054				
01C		204D				
01D		8000				
01E		0029				
01F		104A				
020		4055				
021		204D				
022		8000				
023		0029				
024		104B				
025		6000				
026		104C				
027		6000				
028		9000				
029		0000				
02A		104D				
02B		204A				
02C		002F				
02D		0035				
02E		C029				
02F		0000				

Symbol  
table

Note: It is a common practice for programmers to put the bit string 0xdeadcode into memory locations that are empty or unused. The purpose of this is to help programmers distinguish between unused spaces from used spaces that are initialized to value 0.

Insert symbols into the label column according to the addresses listed in the symbol table. Then erase the addresses in the assembly code.

### Pass (2)

Symbol  
table

Address	Label	Machine code	Assembly code (best guess)	Hash to
000		003F	JNS	S1
001		104A	LOAD	S2
002		404F	SUBT	S3
003		204D	STORE	S4
004		8000	SKIPCOND 000	
005		0029		
006		104A		
007		4050		
008		204D		
009		8000		
00A		0029		
00B		104A		
00C		4051		
00D		204D		
00E		8000		
00F		0029		
010		104A		
011		4052		
012		204D		
013		8000		
014		0029		
015		104A		
016		4053		
017		204D		
018		8000		
019		0029		
01A		104A		
01B		4054		
01C		204D		
01D		8000		
01E		0029		
01F		104A		
020		4055		
021		204D		
022		8000		
023		0029		
024		104B		
025		6000		
026		104C		
027		6000		
028		9000		
029		0000		
02A		104D		
02B		204A		
02C		002F		
02D		0035		
02E		C029		
02F		0000		

[illegible]

## Symbol table Page 2

Note: It is a common practice for programmers to put the bit string 0xdeadcode into memory locations that are empty or unused. The purpose of this is to help programmers distinguish between unused spaces from used spaces that are initialized to value 0.