THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

Compsci210 tutorial

Introduction to Assembly and LC-3 Simulator

Tutorial revision

- We have learnt
  - How to install LC-3 simulator
  - Edit LC-3 assembly codes in editor
  - Run simple programs
  - Debug LC-3 by using **Step Over** button
- This tutorial will cover:
  - Basic LC3 instructions
  - Inputs and Outputs
  - Branching for IF-ELSE, FOR loop, WHILE loop
  - Subroutines

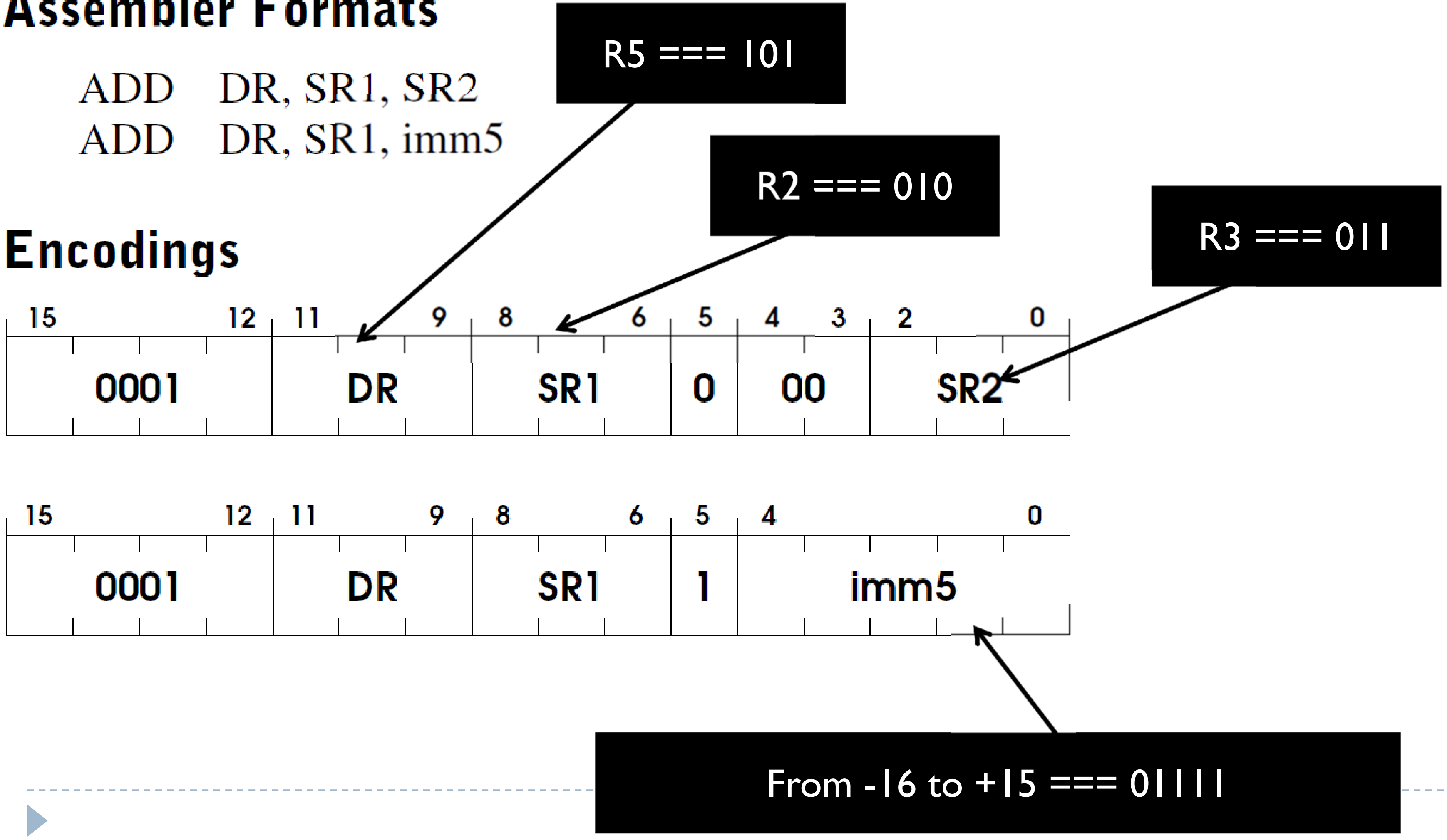
ADD

Similar to AND

Assembler Formats

ADD DR, SR1, SR2
ADD DR, SR1, imm5

Encodings



15–12	11–9	8–6	5	4–3	2–0
0001	DR	SR1	0	00	SR2

15–12	11–9	8–6	5	4–0
0001	DR	SR1	1	imm5

R5 === 101

R2 === 010

R3 === 011

From -16 to +15 === 01111

LD

Assembler Format

LD DR, LABEL

Encoding

15–12	11–9	8–0
0010	DR	PCoffset9

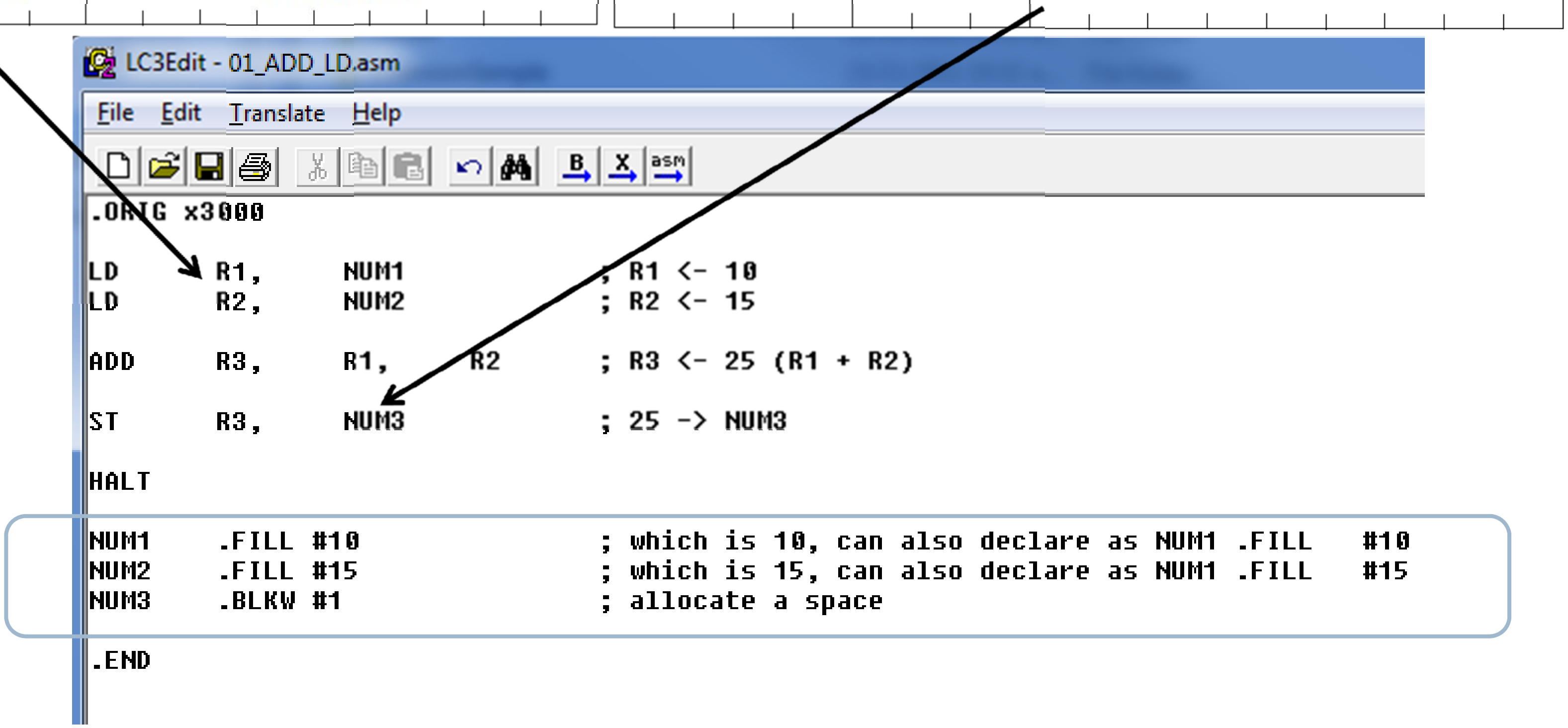
ST

Assembler Format

ST SR, LABEL

Encoding

15–12	11–9	8–0
0011	SR	PCoffset9



```
.ORIG x3000

LD      R1,     NUM1            ; R1 <- 10
LD      R2,     NUM2            ; R2 <- 15

ADD     R3,     R1,     R2      ; R3 <- 25 (R1 + R2)

ST      R3,     NUM3            ; 25 -> NUM3

HALT

NUM1    .FILL #10               ; which is 10, can also declare as NUM1 .FILL   #10
NUM2    .FILL #15               ; which is 15, can also declare as NUM1 .FILL   #15
NUM3    .BLKW #1                ; allocate a space

.END
```

Others: LDR, STR, LDI, STI

BR

Conditional Branch

Assembler Formats

Always in nzp order

BRn	LABEL	BRzp	LABEL
BRz	LABEL	BRnp	LABEL
BRp	LABEL	BRnz	LABEL
BR†	LABEL	BRnzp	LABEL

To Implement IF ELSE LOOP

Encoding

15 12	11	10	9	8 0
0000	n	z	p	PCoffset9

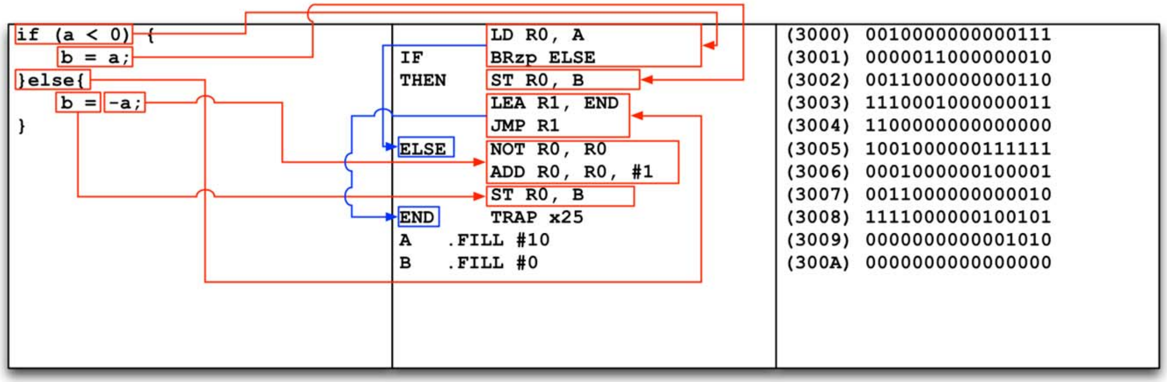
How far from current program counter to the LABEL address

So if LABEL is next line === 000000000

Branch operation

- By getC and Out, you can input 1 character and output 1 character at a time. In order to input and output more, you need loops.
- Loops can be created by using Br (branch operation)
- BR {n|z|p} Label
- BRn branch to Label if register is negative
- BRz branch to Label if register is zero
- BRp branch to Label if register is positive
- → BRzp, BRzn, BRpn…
- BRnzp branch without any condition
- Clearer explanation: http://www.lc3help.com/tutorials/Basic_LC-3_Instructions

An If-Then-Else block

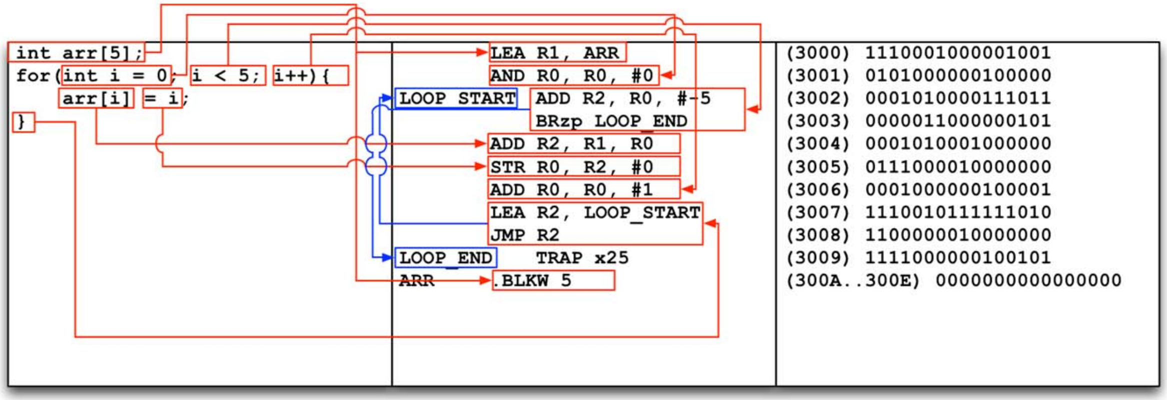


```
if (a < 0) {
    b = a;
}else{
    b = -a;
}
```

```
         LD R0, A
IF       BRzp ELSE
THEN     ST R0, B
         LEA R1, END
         JMP R1
ELSE     NOT R0, R0
         ADD R0, R0, #1
         ST R0, B
END      TRAP x25
A    .FILL #10
B    .FILL #0
```

```
(3000)  0010000000000111
(3001)  0000011000000010
(3002)  0011000000000110
(3003)  1110001000000011
(3004)  1100000000000000
(3005)  1001000000111111
(3006)  0001000000100001
(3007)  0011000000000010
(3008)  1111000000100101
(3009)  0000000000001010
(300A)  0000000000000000
```

A For loop block

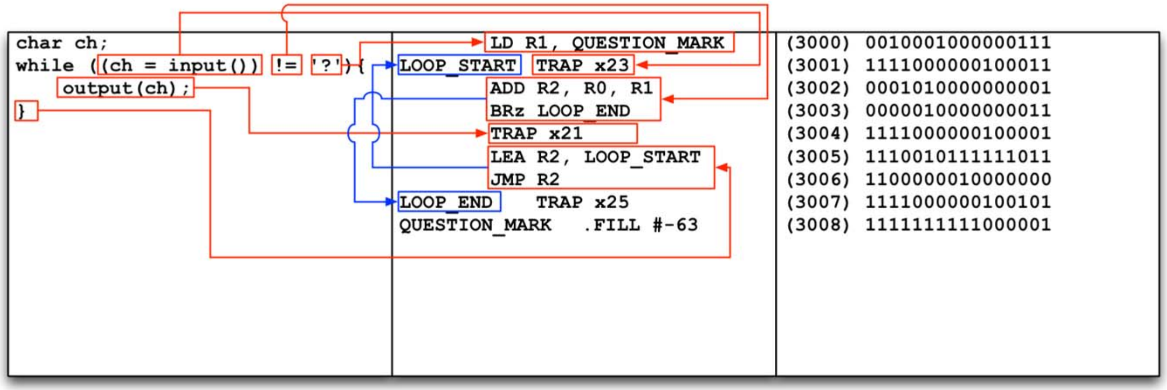


```
int arr[5];
for(int i = 0; i < 5; i++){
    arr[i] = i;
}
```

```
         LEA R1, ARR
         AND R0, R0, #0
LOOP_START  ADD R2, R0, #-5
            BRzp LOOP_END
         ADD R2, R1, R0
         STR R0, R2, #0
         ADD R0, R0, #1
         LEA R2, LOOP_START
         JMP R2
LOOP_END    TRAP x25
ARR       .BLKW 5
```

```
(3000) 1110001000001001
(3001) 0101000000100000
(3002) 0001010000111011
(3003) 0000011000000101
(3004) 0001010001000000
(3005) 0111000010000000
(3006) 0001000000100001
(3007) 1110010111111010
(3008) 1100000010000000
(3009) 1111000000100101
(300A..300E) 0000000000000000
```

A While loop block

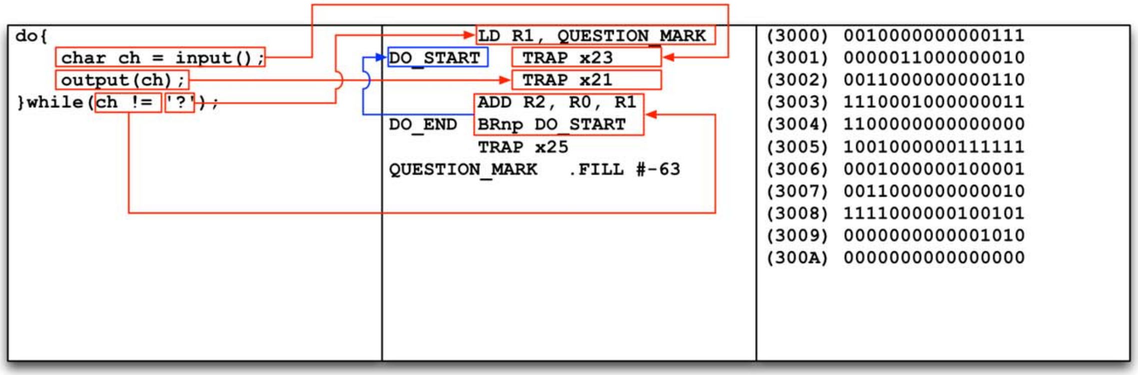


```
char ch;
while ((ch = input()) != '?'){
    output(ch);
}
```

```
            LD R1, QUESTION_MARK
LOOP_START  TRAP x23
            ADD R2, R0, R1
            BRz LOOP_END
            TRAP x21
            LEA R2, LOOP_START
            JMP R2
LOOP_END    TRAP x25
QUESTION_MARK   .FILL #-63
```

```
(3000)  0010001000000111
(3001)  1111000000100011
(3002)  0001010000000001
(3003)  0000010000000011
(3004)  1111000000100001
(3005)  1110010111111011
(3006)  1100000010000000
(3007)  1111000000100101
(3008)  1111111111000001
```

A Do-While Loop block

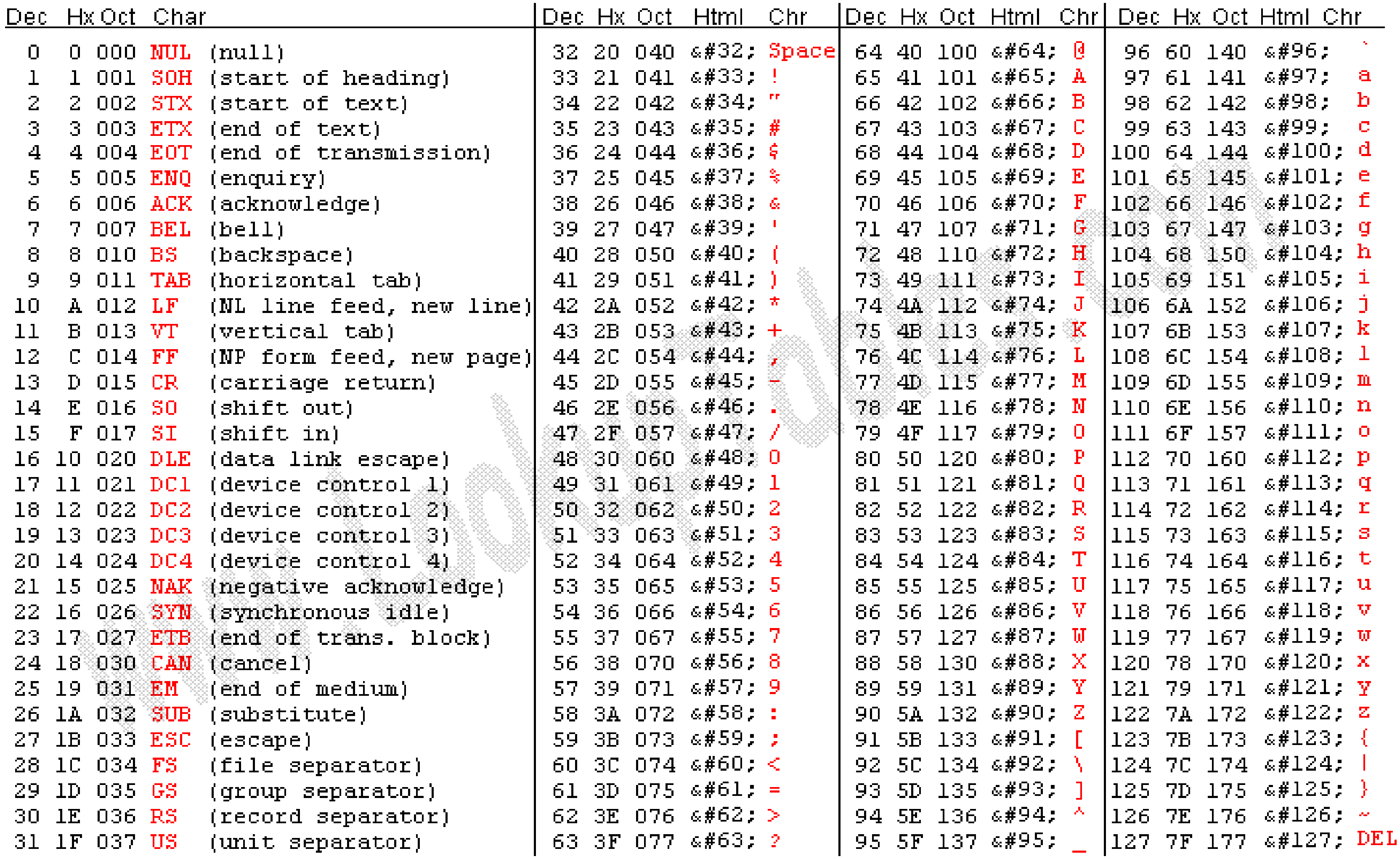


```
do{
    char ch = input();
    output(ch);
}while(ch != '?');
```

```
        LD R1, QUESTION_MARK
DO_START    TRAP x23
            TRAP x21
        ADD R2, R0, R1
DO_END  BRnp DO_START
        TRAP x25
QUESTION_MARK   .FILL #-63
```

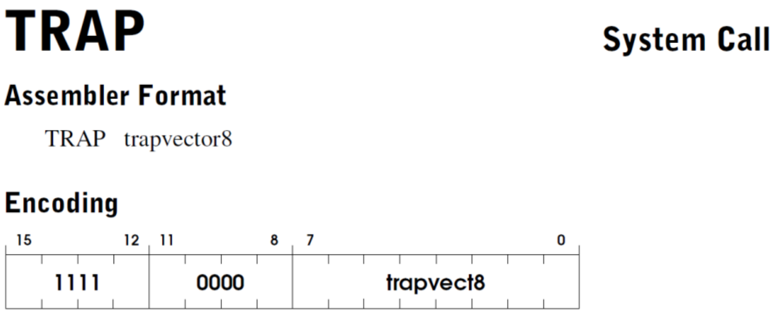
```
(3000)  0010000000000111
(3001)  0000011000000010
(3002)  0011000000000110
(3003)  1110001000000011
(3004)  1100000000000000
(3005)  1001000000111111
(3006)  0001000000100001
(3007)  0011000000000010
(3008)  1111000000100101
(3009)  0000000000001010
(300A)  0000000000000000
```

ASCII standard to encode characters



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Input/output(1)



TRAP **System Call**

Assembler Format

TRAP trapvector8

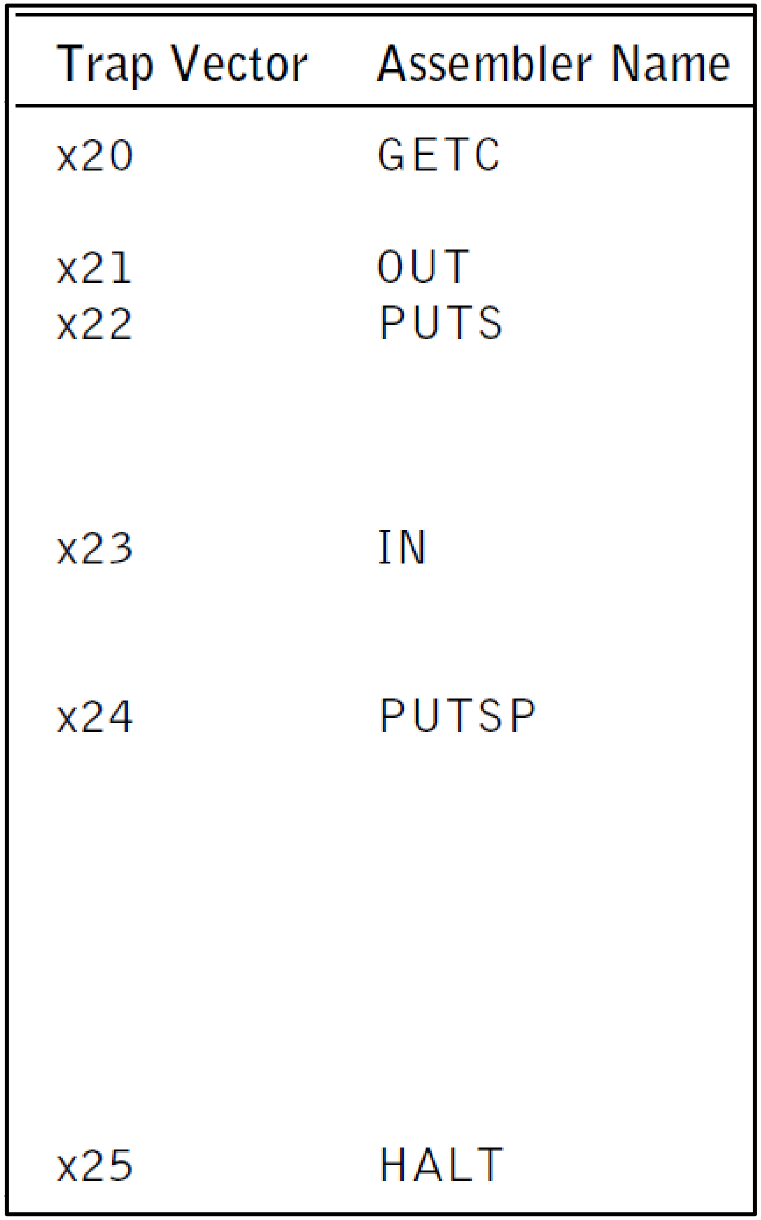
Encoding

15 12	11 8	7 0
1111	0000	trapvect8

- Input and output:
  - Get characters from keyboard to memory/register
  - Print characters from memory/register to screen
- Try running GetC.asm
  - Program does: get 1 input from keyboard and print that out to screen.
- Operations for input/output can be used:
  - Getc
  - Out
  - In
  - Puts

Input/output(2)

- GetC
  - It takes a character from keyboard
  - Store it in Register R0 (ascii value)
- Out
  - It takes ascii value stored in R0
  - Print the correspondent character out to screen
- In
  - It prints out a line ask user to input
  - It takes a character from keyboard
  - Store it in Register R0 (ascii value)
- Puts
  - It prints out a String
-
  - Look at printString.asm

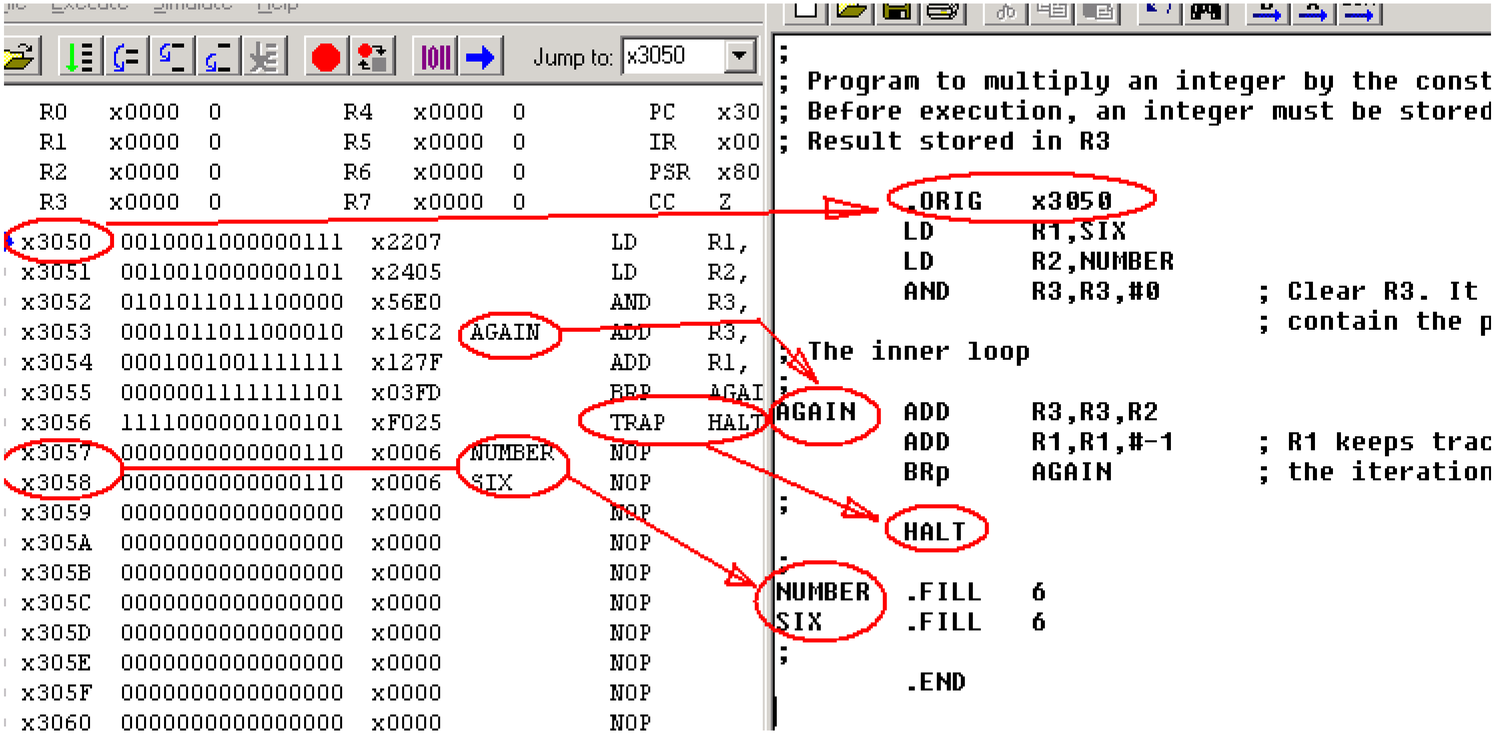


Trap Vector	Assembler Name
x20	GETC
x21	OUT
x22	PUTS
x23	IN
x24	PUTSP
x25	HALT

Quick Demo:
Chapter 7.1 example (1)

- chapter7_code: 7.1.asm
- What s the program doing?
  - Program multiplies an integer by the constant 6.
  - Before execution, an integer must be stored in NUMBER.
  - Result stored in R3
- Operations used:
  - Ld $(register),VariableName ;load value to register from memory
  - And $(register), $(register), #(decimalNumber) ;bitwise operation
  - BRp Label ;branch (goto) to a Label in memory if register is positive

7.1 example (2)



```
 R0    x0000   0          R4    x0000   0          PC    x30
 R1    x0000   0          R5    x0000   0          IR    x00
 R2    x0000   0          R6    x0000   0          PSR   x80
 R3    x0000   0          R7    x0000   0          CC    Z
x3050  0010001000000111  x2207               LD     R1,
x3051  0010010000000101  x2405               LD     R2,
x3052  0101011011100000  x56E0               AND    R3,
x3053  0001011011000010  x16C2  AGAIN        ADD    R3,
x3054  0001001001111111  x127F               ADD    R1,
x3055  0000001111111101  x03FD               BRP    AGAI
x3056  1111000000100101  xF025               TRAP   HALT
x3057  0000000000000110  x0006  NUMBER       NOP
x3058  0000000000000110  x0006  SIX          NOP
x3059  0000000000000000  x0000               NOP
x305A  0000000000000000  x0000               NOP
x305B  0000000000000000  x0000               NOP
x305C  0000000000000000  x0000               NOP
x305D  0000000000000000  x0000               NOP
x305E  0000000000000000  x0000               NOP
x305F  0000000000000000  x0000               NOP
x3060  0000000000000000  x0000               NOP
```

```
;
; Program to multiply an integer by the const
; Before execution, an integer must be stored
; Result stored in R3

        .ORIG   x3050
        LD      R1,SIX
        LD      R2,NUMBER
        AND     R3,R3,#0        ; Clear R3. It
                                ; contain the p
;
; The inner loop
;
AGAIN   ADD     R3,R3,R2
        ADD     R1,R1,#-1       ; R1 keeps trac
        BRp     AGAIN           ; the iteration
;
        HALT
;
NUMBER  .FILL   6
SIX     .FILL   6
;
        .END
```

Exercise 1

- Use LC3 Assembly Instruction set table to convert the following code to Binary ISA codes:

- AND R0,R0,#0
- NOT R0,R0
- AND R1,R1,#0
- ADD R1,R1,#1
- NOT R1,R1
- NOT R1,R1
- ADD R0,R0,R1

Exercise 2

- Use LC3 Assembly Instruction set table to convert the following code to Binary ISA codes:

- LD R2, Num1
- LD R3, Num2
- ADD R4, R2, R3
- HALT
- Num1 .FILL 5
- Num2 .FILL 6

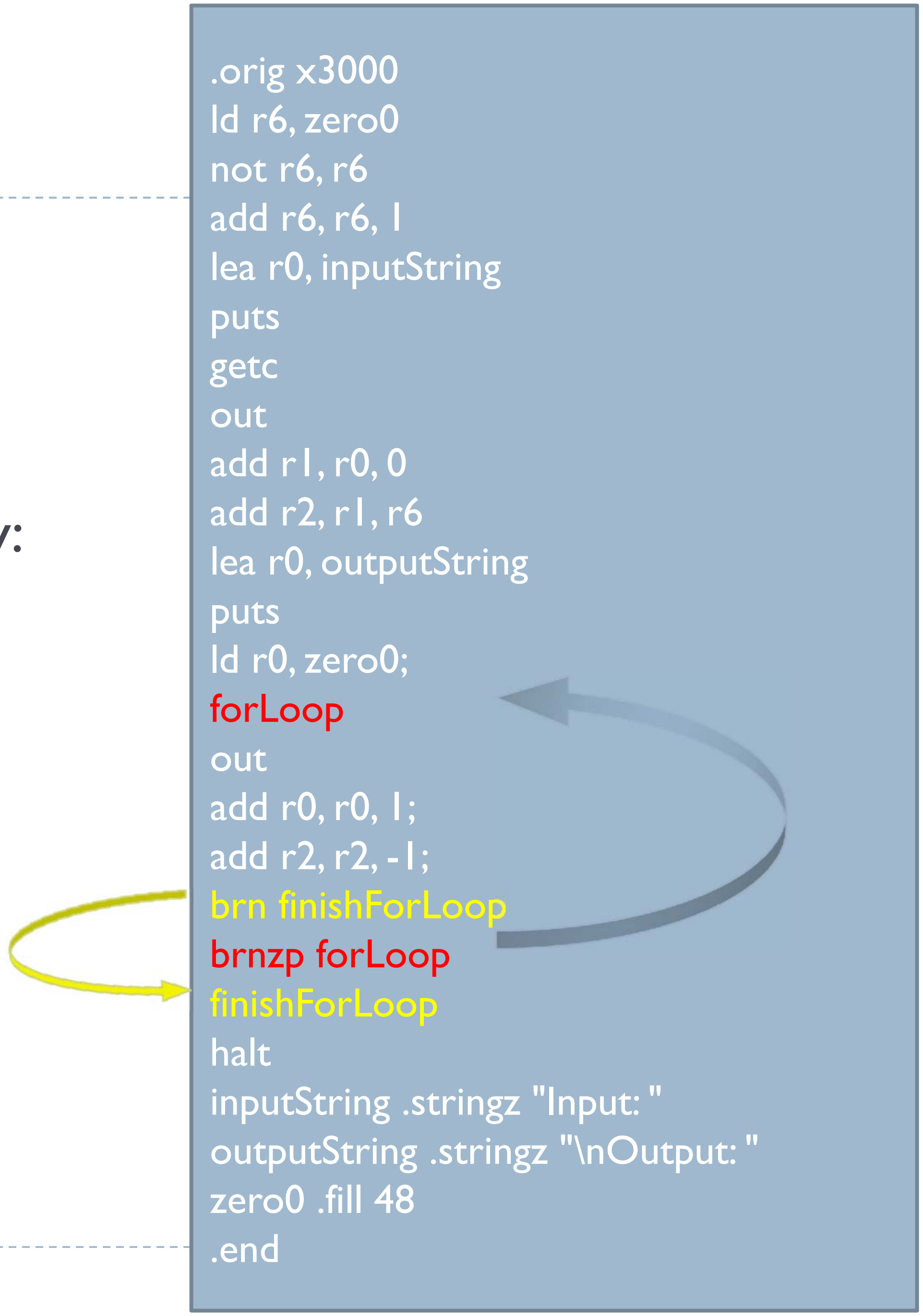
- What do the codes do?

Exercise 3 and 4

- Do exercises:
  - Input a number from 0 to 9
  - Print out all the number from 0 to that number
  - Example:
    - Input: 4
    - Output: 0 1 2 3 4
- Create an example to echo an user input, i.e:
  - Hi, what is your name?
  - George Alexander Louis
  - Hi George Alexander Louis, nice to meet you.

Exercise 3 answer

- Steps need to complete:
  - Get input as a character
  - Turn that character to int by:
    - take away offset (‘0’)
    - N = ‘5’ – ‘0’
  - Make a for loop to print:
    - N times.
  - Start from
    - ‘0’
  - Use BR wisely



```
.orig x3000
ld r6, zero0
not r6, r6
add r6, r6, 1
lea r0, inputString
puts
getc
out
add r1, r0, 0
add r2, r1, r6
lea r0, outputString
puts
ld r0, zero0;
forLoop
out
add r0, r0, 1;
add r2, r2, -1;
brn finishForLoop
brnzp forLoop
finishForLoop
halt
inputString .stringz "Input: "
outputString .stringz "\nOutput: "
zero0 .fill 48
.end
```