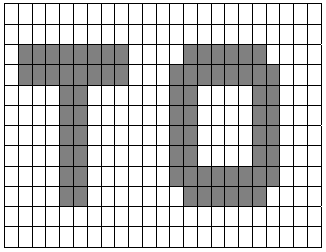
RLC

(Run Length Coding)

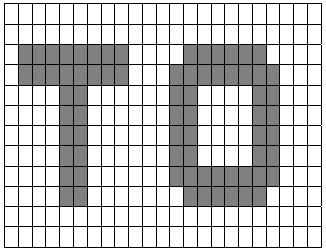
Print and use
the specification document
on the class web site

See syllabus for assignment type
individual or team

There is data that consist of just
runs of black and white picture elements
(scanned text, fax data, line drawings)

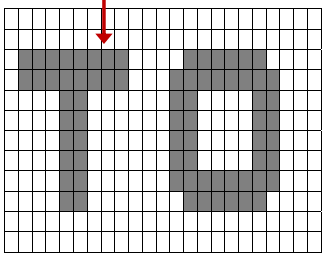


Run Length Coding is data compression
in which common runs of data are
stored as a data value (black or white)
and a count of the run length



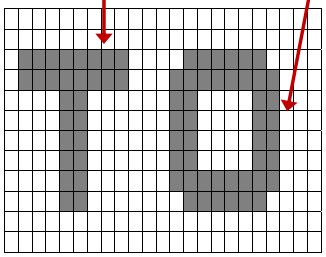
Run Length Coding is data compression
in which common runs of data are
stored as a data value (black or white)
and a count of the run length

Top of T = 8 black pels



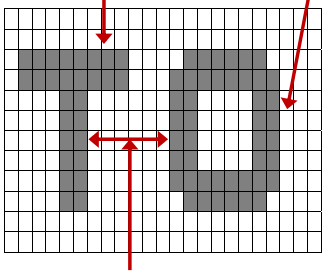
Run Length Coding is data compression
in which common runs of data are
stored as a data value (black or white)
and a count of the run length

Top of T **Side of O = 2 black pels**



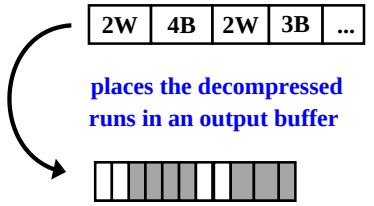
Run Length Coding is data compression
in which common runs of data are
stored as a data value (black or white)
and a count of the run length

Top of T **Side of O**



Space between letters 6 white pels

You will write a subroutine RLC
that is linked with a C main program
that decompresses Run Length Codes



places the decompressed
runs in an output buffer

You will write a subroutine RLC that is linked with a C main program that decompresses Run Length Codes

Input

2W	4B	2W	3B	...

- pointer input string

places the decompressed runs in an output buffer

- pointer output buffer

Codes are 4 bits packed two per byte

RLC	Meaning
0000	run length 0 of current color
0001	run length 1 of current color
0010	run length 2 of current color
0011	run length 3 of current color
	...
1110	run length 14 of current color
1111	run of current color that goes to the end of the 80 byte line

A byte of zero signifies end of the input

Feature	Meaning
0000 0000	End of data Return to caller

Rules for decompressing

- Each pel is 1 byte wh=20h bl=DBh
- The only items you place in the output buffer are wh or bl pels
- Line length is 80 pels
- First run on a line is assumed white
- Runs alternate wh bl wh bl ...
- The last run for a line is 1111
- All data is valid ... no error checking

Why do you need a run length of 0?

- First run on a line is assumed white. If it must be black then you to first need a run of white with zero length
- If a run is > 14 (e.g. 20) then it must be sent as (assume bl run)
  - run of 14 bl
  - run of 0 wh
  - run of 6 bl

This is the data the letter T.
In hex it consists of these 26 runs ...

F0 6F 06 F2 2F 22 F2 2F 22 F2 2F 22 FF

Which produce this image ....

F=wh run to end of line	
0=wh run 0 6=bl run 6 F=wh run eol	
0=wh run 0 6=bl run 6 F=wh run eol	
2=wh run 2 2=bl run 2 F=wh run eol	
2=wh run 2 2=bl run 2 F=wh run eol	
2=wh run 2 2=bl run 2 F=wh run eol	
2=wh run 2 2=bl run 2 F=wh run eol	
2=wh run 2 2=bl run 2 F=wh run eol	
2=wh run 2 2=bl run 2 F=wh run eol	
F=wh run to end of line	

RLC Subroutine Description

rlc (inlist, outlist);

```
_rlc:
   push bp          ;save bp
   mov  bp,sp       ;point to stack
   push si          ;save si
   push di          ;save di
   mov  si,[bp+4]   ;si pts to input
   mov  di,[bp+6]   ;di pts to output
;---------------------------------

      Your rlc code goes here

;---------------------------------
exit:
   pop  di          ;restore di
   pop  si          ;restore si
   pop  bp          ;restore bp
   ret              ;return
```

RLC Subroutine Description

rlc (inlist, outlist);

_rlc:
push bp ;save bp

Remember ... your subroutines does *not* do any input or output

All data is passed in memory

The driver creates both lists ... you fill in the output list

pop bp ;restore bp
ret ;return

Step 1. Create a design

We provide a working C program as a guide

Functional but not optimized for performance

Step 2. Code your solution

Retrieve the grading system packed in a self-extracting file named *unpack.exe*.

To unpack it type: *unpack*

- ***rlc.m***
  the model for your subroutine
- **rename it *rlc.asm***
  add your code to that file
- ***rlcdrvr.obj***
  testing and grading driver
- **link *rlcdrvr.obj* with *rlc.obj***

Step 3. Test and debug your solution.

Use the 3 tests built in to the driver program to test your rlc subroutine

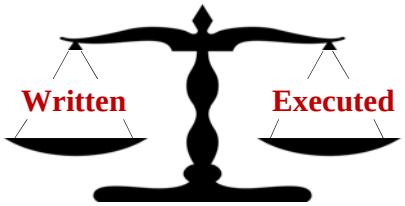
To run a test, type: *testrlc #*

The output will go the display

Step 4. Grade

Type: *gradrlc*

- **40 points correct answers**
- **20 points number of instrs *written***
- **20 points number of instrs *executed***
- **20 points documentation**



Step 5. Submit your assignment

Electronically submit the file

rlc.ans

created by the grading system

Intel architected a number of very powerful string instructions that facilitate operations on sequences of bytes or words

They use implied operands and are thus compact and fast

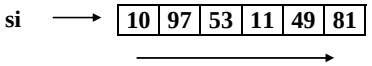
RLC's input and output are strings

Using string instructions can be useful

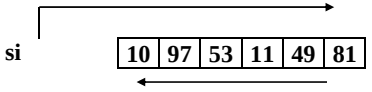
(Class Notes Chapter 16A)

Load accumulator from a string

Low to high
- **si points to the beginning of the string**
- **process from beginning to end**
- **clear the direction flag**



High to low
- **si points to the end of the string**
- **process from end to beginning**
- **set the direction flag**

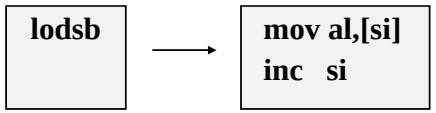


Load accumulator from a string

- **Initialization**

Execute *cld* so that indices increase
Load si with a pointer to the input string

- **Execution**

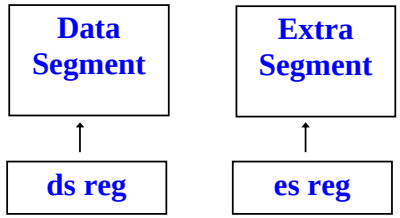


Cuts instructions
for accessing the input string

Store accumulator into a string

- **Initialization**

Execute *cld* so that indices increase
Load di with a pointer to the output string

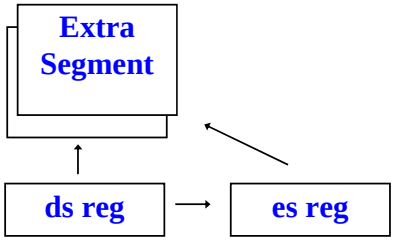


Store acc into string
works with the extra segment

Store accumulator into a string

- **Initialization**

Execute *cld* so that indices increase
Load di with a pointer to the output string



Load es with ds so
Extra Segment and Data Segment
are the same

Store accumulator into a string

- **Initialization**

Execute *cld* so that indices increase
Load di with a pointer to the output string
Set the es register to the value in ds
Load cx with the number of bytes to store

- **Execution**

```
rep  stosb
```

```
while cx != 0
x: mov es:[di],al
   inc   di
   loop x:
```

Cuts instructions
for storing in the output string

Net Effect For RLC

Implementation	Instructions Executed
Convert the C logic to assembler	
Modify the C logic to use lodsb and stosb	

- **In general look for better ways to implement C statements.**

For example ... find a better way than this to alternate output colors

```
if (cur==wh) cur=bl; else cur=wh;
```