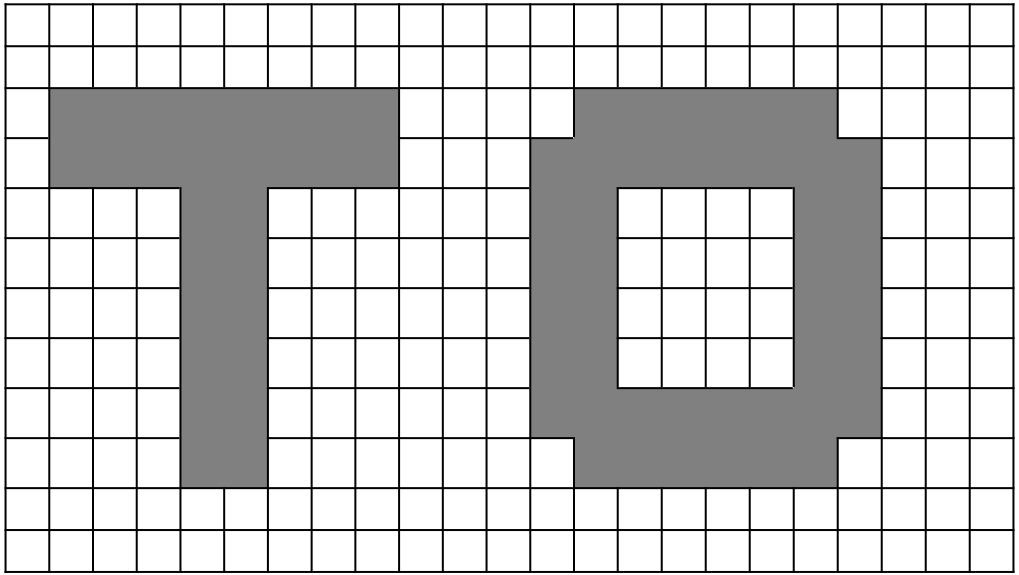
RLC

CSC 236

RLC — run length coding

- Read the document
- Optional team assignment
  - Can work with one other student (2-person team)
  - Both must submit assignment
- Run-length encoding
  - Data compression
  - B/W image consists of rows
  - Each row has black or white cells
  - Can be encoded as repeated set of
    - Run of white cells then
    - Run of black cells

- Row 3
  - 1W
  - 8B
  - 4W
  - 6B
  - 4W
- Row 9
  - 4W
  - 2B
  - 6W
  - 8B
  - 3W



Task

- Write subroutine
  - Linked with C main program
  - Decompresses RLC
- Input
  - Pointer to string — run lengths
  - Pointer to buffer — decompressed runs
  - Each buffer holds 80 characters/bytes
  - Each row is 80 columns wide

Code

- 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 to end of line (80)

Feature	Meaning
0000 0000	End of data Return to caller

Rules

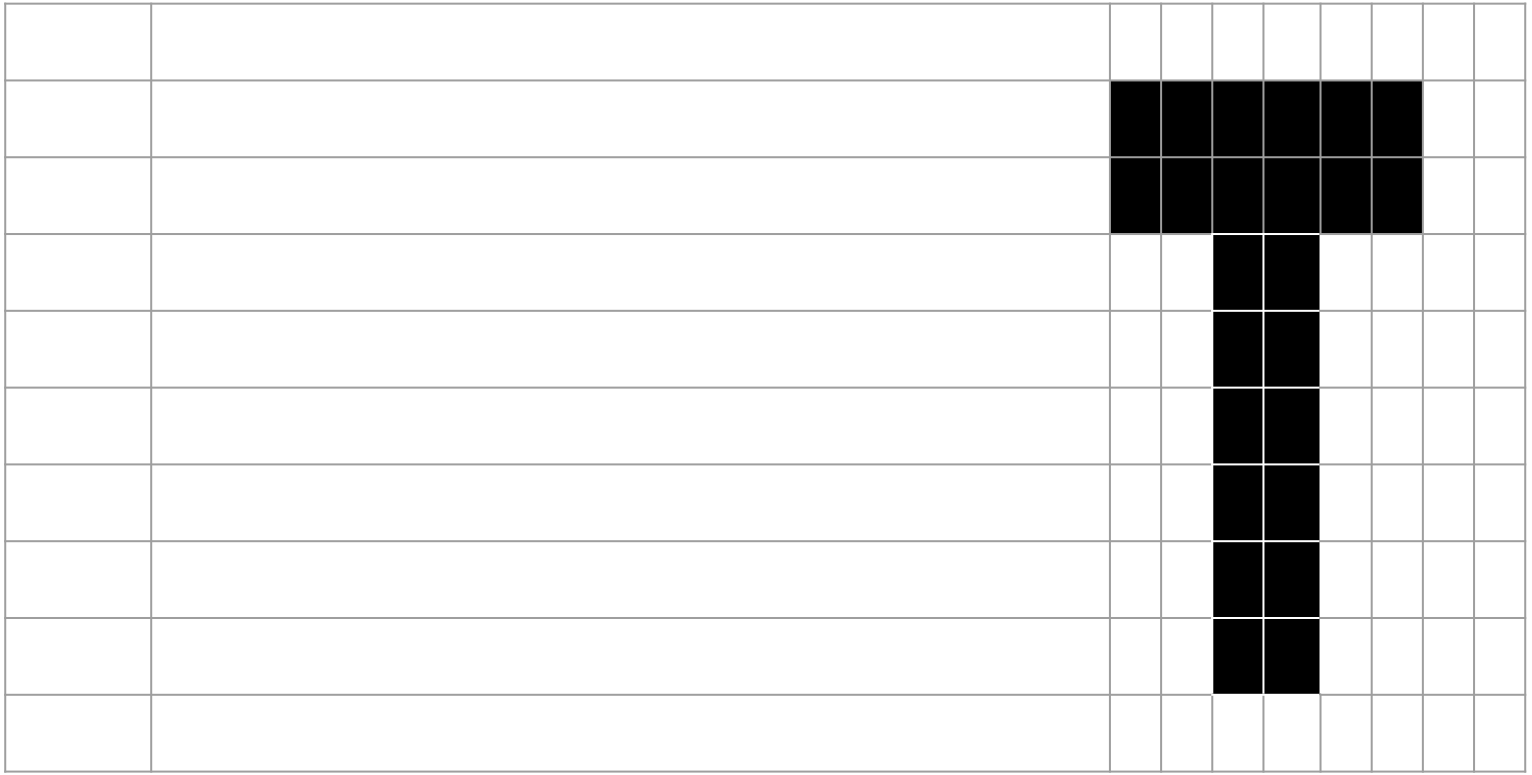
- Each pixel is one byte
  - White = 20h
  - Black = DBh
- Only put white or black in output buffer
- Line length is 80 pixels (bytes)
- First run on each line is white
- Runs alternate between w, b, w, b, …
- Color is not in data (it is assumed by the algorithm)
- Last run for a line is 1111 — fill to column 80
- No error checking required — data can be assumed valid

Run length of 0

- Why support a run length of zero?
- Suppose the first pixel is black
  - First run is white by definition
  - Need a run of 0 for white
- Suppose run is more than 14 (say, 20)
  - 1110 (black)
  - 0000 (white)
  - 0110 (black)

Letter T

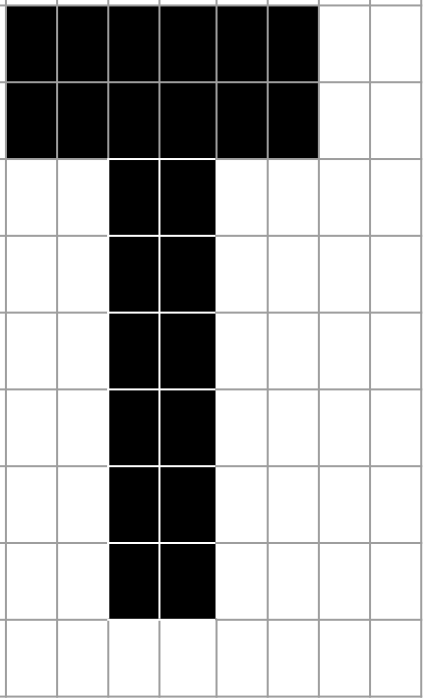
F0 6F 06 F2 2F 22 F2 2F 22 F2 2F 22 FF



Letter T

E0 6F 06 F2 2F 22 F2 2F 22 F2 2F 22 FF

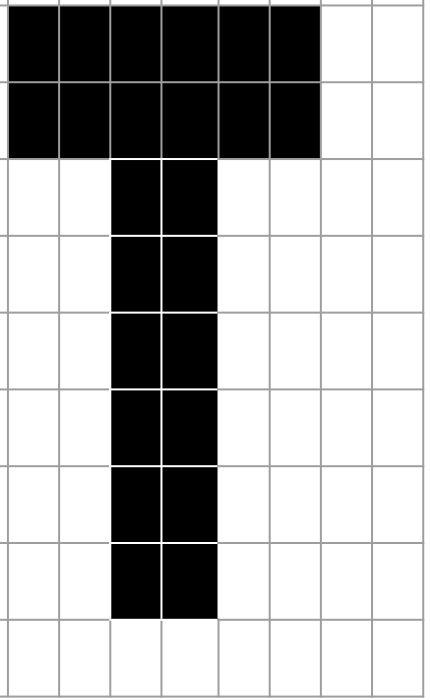
F	wEOL (w **to end)**



Letter T

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

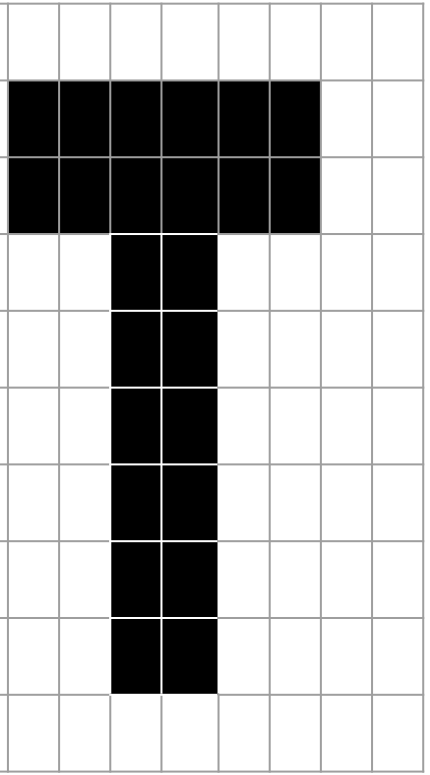
F	wEOL (w **to end)**	
06F	w0 b6 wEOL	



Letter T

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

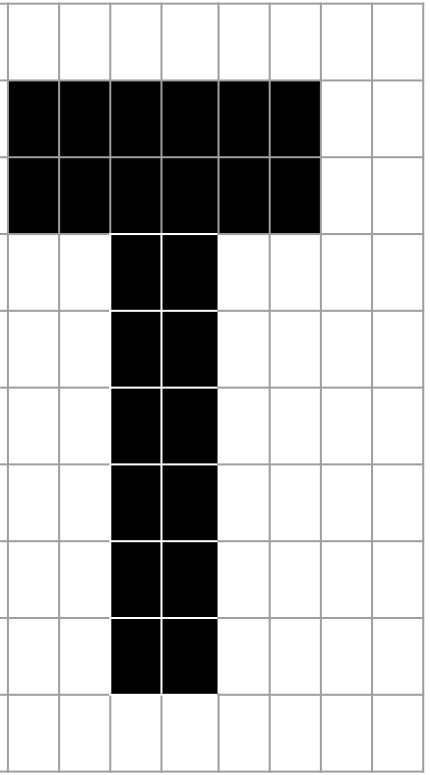
F	wEOL (w **to end)**
06F	w0 b6 wEOL
06F	w0 b6 wEOL



Letter T

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

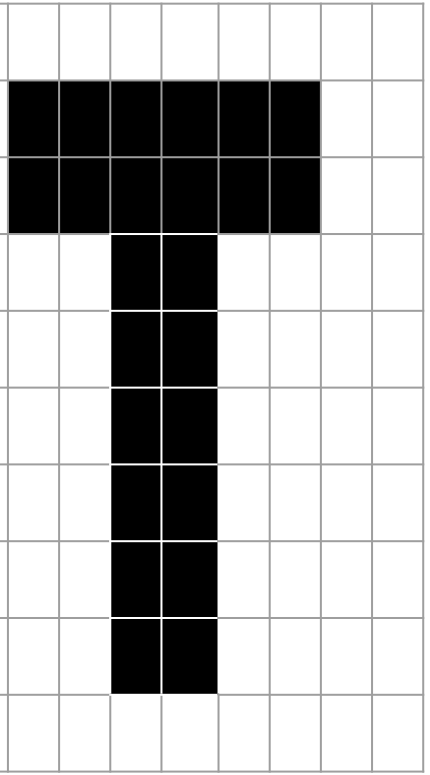
F	wEOL (w **to end)**
06F	w0 b6 wEOL
06F	w0 b6 wEOL
22F	w2 b2 wEOL



Letter T

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

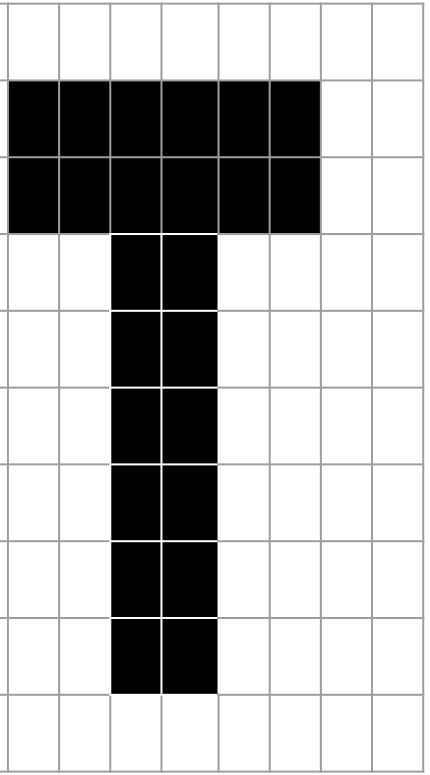
F	wEOL (w **to end)**
06F	w0 b6 wEOL
06F	w0 b6 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL



Letter T

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

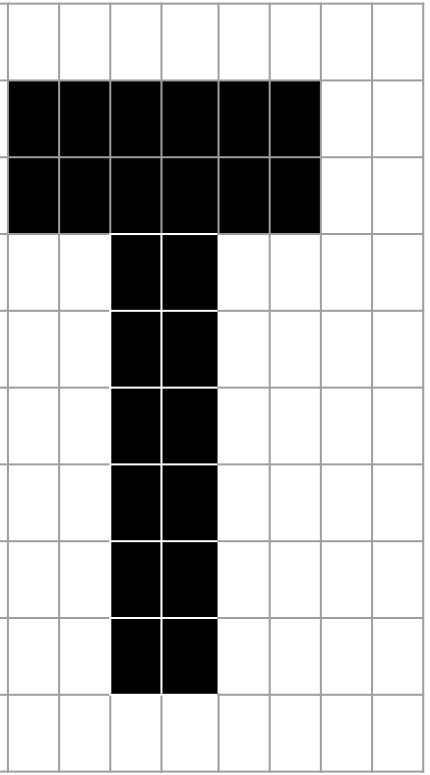
F	wEOL (w **to end)**
06F	w0 b6 wEOL
06F	w0 b6 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL



Letter T

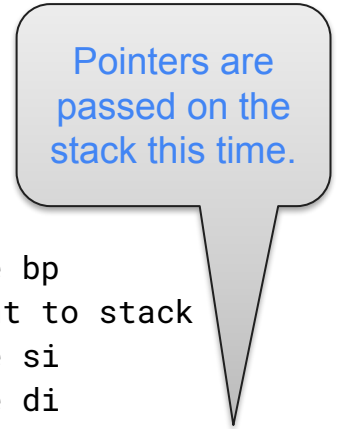
F0 6F 06 F2 2F 22 F2 2F 22 F2 2F 22 FE

F	wEOL (w **to end)**
06F	w0 b6 wEOL
06F	w0 b6 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL
22F	w2 b2 wEOL
F	wEOL



Guidance

- C linkage code provided
- Subroutine does not do input or output
- All image data are passed in memory
- Driver creates both buffers
- Subroutine fills in the output buffer




```
_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
```

Steps

- Create a design
  - Working C program
  - Guide
  - Functional; not optimal
- Download
  - Download *unpack.exe*
  - Extract in DOSbox, in the RLC subdirectory.

Steps

- Resources
  - rlc.m — model for subroutine (C linkage)
  - Rename rlc.asm
  - rlcdrvr.obj — testing and grading driver
- Executable
  - Assemble rlc.asm — ml /c /Zi /Fl rlc.asm
  - Link rlcdrvr.obj with rlc.obj — link /CO rlcdrvr.obj rlc.obj
- Test
  - testrlc <n> # where n=1, 2, or 3

Steps

- Grade
  - `graderlc`
  - 40% — correct answers
  - 20% — instructions written
  - 20% — instructions executed
  - 20% — documentation
- Submit
  - rlc.ans

Design ideas

- Intel string instruction
  - CISC instructions
  - Powerful
  - Designed for word processing applications
  - RLC input & output are strings
  - See Class Notes Chapter 16A

lodsb

- Load accumulator from string
  - Loads al with next item in a string
  - si points to next item in string
  - Updated si
- Initialization
  - Execute `cld` — clear direction flag (`std` set it)
  - Load si with base of string
- `lodsb`
  - `mov al,[si]`
  - `inc si`

stosb

- Store accumulator from string
  - Stores al into string
  - di points to location
  - Updates di
  - Data in extra segment
- Initialization
  - Execute `cld`
  - Load di with base of string
  - Set es register to value in ds
- `stosb`
  - `mov es:[di],al`
  - `inc di`

- Output several bytes
  - Mov byte (black or white) into al
  - Load count into cx
  - Execute `rep stosb`
- Essentially
  - ```
    while (cx!=0) {
      mov es:[di],al
      inc di
      dec cx
    }
    ```
    ```

Alternating colors

- Straightforward
  - if (cur==W) cur=B else cur=W
- Find a better way to alternate colors