Characters, Bytes, Halfwords

```
.data
.word  5,3,-8
.byte  5, 3, -8
.ascii  "hello_world"   <- 11 characters ≡ 11 bytes
.asciz  "hello_world"_  <- 12 bytes
                           adds byte with value ∅ at end
                              ∅ ≡ NULL in ASCII
                           we call this a null-terminated string
.hword  5, 3, -8             the NULL character is not printable
.end
        we'll use this for audio data
```

characters

ASCII - American Standard Code for Information Interchange
- international standand adopted by ISO to map characters to byte values
- values 0-127, 7bits

other codes: EBCDIC (old IBM standard)
Unicode (newer standard for international languages, uses 16b per character)

character	value
'∅'	48
'1'	49
'A'	65
'B'	66

can use '∅' anywhere you would use the value 48

etc - see Wikipedia

.byte 5, 3, -8, '0', 48

byte value 48 appears twice

$\left.\begin{array}{ll}\text{ldw} & \text{r2, }\underline{\text{0(r16)}}\\ \text{stw} & \text{r2, }\underline{\text{0(r16)}}\end{array}\right\}$ effective address (e.a) must be multiple of 4

$\left.\begin{array}{ll}\text{ldh} & \text{r2, 0(r16)}\\ \text{sth} & \text{r2, 0(r16)}\\ \underline{\text{ldhu}} & \text{r2, 0(r16)}\end{array}\right\}$ e.a. must be multiple of 2

ldh reads a halfword (16b) from memory and puts it as a 32b signed value into a register - upper 16b are sign-extended

ldhu for <u>unsigned data</u> - upper 16b are zero

$\left.\begin{array}{ll}\text{ldb} & \text{r2, 0(r16)}\\ \text{stb} & \text{"}\\ \text{ldbu} & \text{"}\end{array}\right\}$ any value OK, e.a. can be odd

Aside :- using byte and halfword data makes it possible to place a word at an invalid <u>unaligned</u> address that is not a multiple of 4

eg:
```
    .byte 42
    .word 16243   <-- this word appears after
                      the byte 42, but the
                      word address must be
                      a multiple of 4
```

$\downarrow$

- Nios II assembler automatically corrects this by padding with zeros:

```
    .byte 42
    .byte 0,0,0   <-- extra bytes added
                      by assembler
    .word 16243
```

- sometimes (very rarely) you need to control this using ".align" directive → Google 'gas manual' then search for .align

putting char to terminal

r23 holds IOBASE

```
movi   r4, '0'
~~stwio  r4, TERMINAL (r23)~~
call   putchar
```

use 'call putchar' instead

```
putchar:  ldwio  r2, TERMINALCTL (r23)
          srli   r2, r2, 16
          beq    r2, r0, putchar
          stwio  r4, TERMINAL (r23)
          ret
```

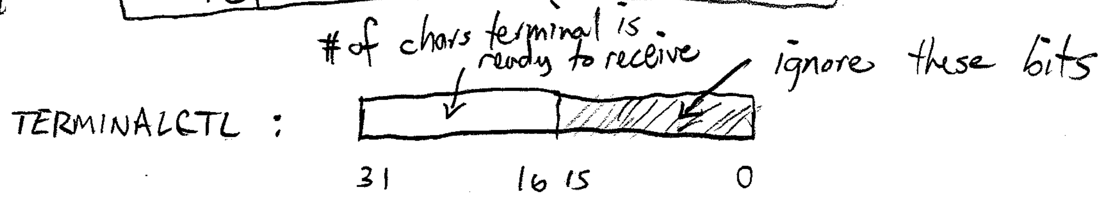
check for space before sending

send it

control info about whether terminal is ready for a new char

2 different locations

send/recv the character

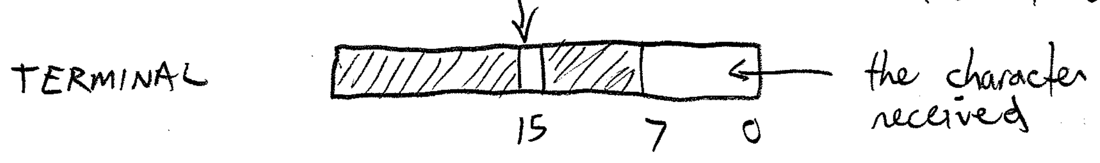


getting char from terminal

```
getchar:  ldwio  r2, TERMINAL (r23)
          andi   r3, r2, 0x8000
          beq    r3, r0, getchar
          andi   r2, r2, 0xff
          ret
```

bit #15: 1 = char avail, 0 = not avail

strip away all other bits (useless)



```
/* Print a null-terminated ASCII string on the terminal using putchar */

.include "ubc-de1media-macros.s"

.global _start

.text
_start:     movia sp, DRAM_END      /* init stack pointer */
            movia r23, IOBASE

            movia r22, string
putmsg:     ldb   r4, 0(r22)        /* get next char to send */
            beq   r4, r0, donesend

            call  putchar           /* send next char */
            addi  r22, r22, 1
            br    putmsg            /* go get next char */

donesend:
echo_kb:    call  getchar           /* get char from user */
            mov   r4, r2
            call  putchar           /* echo it back to user */
            br    echo_kb


/* ************************************************************** */
putchar:    ldwio r2, TERMINALCTL(r23)
            srli  r2, r2, 16
            beq   r2, r0, putchar         /* wait to transmit */
            stwio r4, TERMINAL(r23)       /* send char */
            ret


/* ************************************************************** */
getchar:    ldwio r2, TERMINAL(r23)       /* r2 gets char plus more info */
            andi  r3, r2, 0x8000          /* bit 15 of r2 = 1 means char ready */
            beq   r3, zero, getchar       /* wait to receive */
            andi  r2, r2, 0xff            /* optional: isolate single byte */
            ret

/* ************************************************************** */

.data

string:
.asciz      "Hello, world. Please type a message below.\n"

.end
```