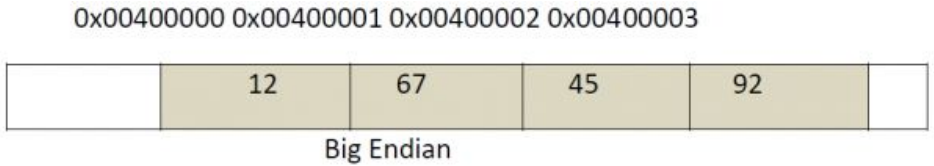
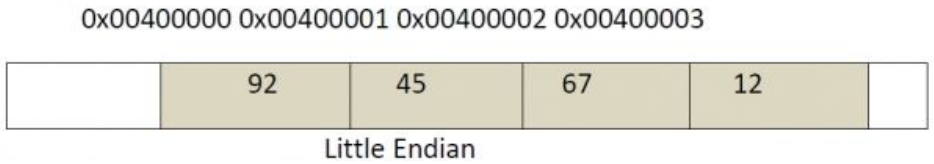
MKFS.C

STEP ONE: GET A DRIVE
STEP TWO: ~
STEP THREE: TADA, A FILE SYSTEM

XINT & XSHORT

- Convert stuff to intel byte order
  - Little endian
  - Least significant bit is stored at lower address
- Representation of 0x12674592





Rsect & wsect

- Go to the “sector” a.k.a offset within the disk image file.
- Read/Write buffer to the sector

Winode & rinode

- Write inode to disk and read inode from disk
- Make use of
  - **INODEOFF** - inode offset
  - **IPB** - offset within block for inode
  - **BSIZE** - block size

IALLOC

- Accept a type
- Increment `**freeinode**` to keep track of number of inodes
- Zero out **dinode**
- Write type
- Write the **dinode** to disk
- Return the **inum**

BALLOC

- Allocate the amount of block indicated by used
- Mark **bitmap** to indicate the blocks allocated are no longer free

IALLOCBLOCKS

- Give an inode some amount of blocks (extent)

IAPPEND

- Add content to the end of the extent of an inode

What is mkfs.c doing?

- Setup the **superblock**
- Allocate root directory
- Add `.`, `..`, `console`
- Loop through the user directory and add every file into the xk file system
- User `**iappend**` to update the extent storing `**dirent**`
- Update the bitmap using `**balloc**`

MACRO

INODEOFF - inode offset

IPB - offset within block for inode

BSIZE - block size

Important variables

`**freeinode**` - tracks the position of the next free inode

`**freeblock**` - tracks the position of the next free block

Debugging

mkfs.c will run during the compiling process

For print debugging:

- In line 49 of **user/Makefrag**, remove the redirection to **/dev/null**

```
45  $(O)/mkfs: mkfs.c
46          $(QUIET_GEN)$(HOST_CC) -I . -o $@ $<
47
48  $(O)/fs.img: $(O)/mkfs $(XK_UPROGS) $(XK_TEXT_FILES)
49          $(QUIET_GEN)$(O)/mkfs $@ $(XK_UPROGS) $(XK_TEXT_FILES) > /dev/null
```