; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;

; UNIX.ASM (RETRO UNIX 8086 Kernel - Only for 1.44 MB floppy disks)

; ----------------------------------------------------------------------------

; U5.ASM (include u5.asm) //// UNIX v1 -> u5.s

; RETRO UNIX 8086 (Retro Unix == Turkish Rational Unix)

; Operating System Project (v0.1) by ERDOGAN TAN (Beginning: 11/07/2012)

; 1.44 MB Floppy Disk

; (11/03/2013)

;

; [ Last Modification: 07/08/2013 ] ;;; completed ;;;

;

; Derivation from UNIX Operating System (v1.0 for PDP-11)

; (Original) Source Code by Ken Thompson (1971-1972)

; <Bell Laboratories (17/3/1972)>

; <Preliminary Release of UNIX Implementation Document>

;

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; 07/08/2013 iget

; 01/08/2013 alloc, (free3, free), itrunc

; 31/07/2013 u.rw -> rw, setimod, mget

; 28/07/2013 iget, icalc (u.rw)

; 21/07/2013 alloc, free, imap

; 18/07/2013 iget

; 17/07/2013 icalc (inode->i), iget

; 09/07/2013 iget (cdev=1)

; 29/04/2013 access modification

; 26/04/2013 imap, iget (mntd->mdev)

; 24/04/2013 access

; 23/04/2013 itrunc

; 07/04/2013 alloc, free, iget, icalc

; 02/04/2013 alloc

; 01/04/2013 alloc

; 24/03/2013 mget

; 22/03/2013 mget

; 11/03/2013

mget:

; 31/07/2013

; 24/03/2013

; 22/03/2013

; Get existing or (allocate) a new disk block for file

;

; INPUTS ->

; u.fofp (file offset pointer)

; inode

; u.off (file offset)

; OUTPUTS ->

; r1 (physical block number)

; r2, r3, r5 (internal)

;

; ((AX = R1)) output

; (Retro UNIX Prototype : 05/03/2013 - 14/11/2012, UNIXCOPY.ASM)

; ((Modified registers: DX, BX, CX, SI, DI, BP))

; mov \*u.fofp,mq / file offset in mq

; clr ac / later to be high sig

; mov $-8,lsh / divide ac/mq by 256.

; mov mq,r2

; bit $10000,i.flgs / lg/sm is this a large or small file

; bne 4f / branch for large file

mget\_0:

mov si, word ptr [u.fofp] ; 24/03/2013

mov bl, byte ptr [SI]+1

xor bh, bh

; BX = r2

test word ptr [i.flgs], 4096 ; 1000h

; is this a large or small file

jnz short mget\_5 ; 4f ; large file

test bl, 0F0h ; !0Fh

; bit $!17,r2

jnz short mget\_2

; bne 3f / branch if r2 greater than or equal to 16

and bl, 0Eh

; bic $!16,r2 / clear all bits but bits 1,2,3

mov ax, word ptr i.dskp[BX] ; AX = R1, physical block number

; mov i.dskp(r2),r1 / r1 has physical block number

or ax, ax

jnz short mget\_1 ; if physical block number is zero

; bne 2f / if physical block num is zero then need a new block

; / for file

call alloc

; jsr r0,alloc / allocate a new block

; AX (r1) = Physical block number

mov word ptr i.dskp[BX], ax

; mov r1,i.dskp(r2) / physical block number stored in i-node

call setimod

; jsr r0,setimod / set inode modified byte (imod)

call clear

; jsr r0,clear / zero out disk/drum block just allocated

mget\_1: ; 2:

; AX (r1) = Physical block number

retn

; rts r0

mget\_2: ; 3: / adding on block which changes small file to a large file

call alloc

; jsr r0,alloc / allocate a new block for this file;

; / block number in r1

; AX (r1) = Physical block number

call wslot

; jsr r0,wslot / set up I/O buffer for write, r5 points to

; / first data word in buffer

; AX (r1) = Physical block number

mov cx, 8 ; R3, transfer old physical block pointers

; into new indirect block area for the new

; large file

mov di, bx ; r5

mov si, offset i.dskp

; mov $8.,r3 / next 6 instructions transfer old physical

; / block pointers

; mov $i.dskp,r2 / into new indirect block for the new

; / large file

xor ax, ax ; mov ax, 0

mget\_3: ;1:

movsw

; mov (r2),(r5)+

mov word ptr [SI]-2, ax

; clr (r2)+

loop mget\_3 ; 1b

; dec r3

; bgt 1b

mov cl, 256-8

; mov $256.-8.,r3 / clear rest of data buffer

mget\_4: ; 1

rep stosw

; clr (r5)+

; dec r3

; bgt 1b

; 24/03/2013

; AX (r1) = Physical block number

call dskwr

; jsr r0,dskwr / write new indirect block on disk

; AX (r1) = Physical block number

mov word ptr [i.dskp], ax

; mov r1,i.dskp / put pointer to indirect block in i-node

or word ptr [i.flgs], 4096 ; 1000h

; bis $10000,i.flgs / set large file bit

; / in i.flgs word of i-node

call setimod

; jsr r0,setimod / set i-node modified flag

jmp short mget\_0

; br mget

mget\_5: ; 4 ; large file

; 05/03/2013 (UNIXCOPY.ASM)

;mov ax, bx ; ax <= 255 for this file (UNIX v1, RUFS) system

;mov cx, 256 ; 01/03/2013 no need a division here

;xor dx, dx ; 01/03/2013 no need a division here

;div cx ; 01/03/2013 no need a division here

;and bx, 1FEh ; zero all bit but 1,2,3,4,5,6,7,8

; gives offset in indirect block

;push bx ; R2

;mov bx, ax ; calculate offset in i-node for pointer

; to proper indirect block

;and bx, 0Eh

;mov ax, word ptr i.dskp[BX] ; R1

; mov $-8,lsh / divide byte number by 256.

; bic $!776,r2 / zero all bits but 1,2,3,4,5,6,7,8; gives offset

; / in indirect block

; mov r2,-(sp) / save on stack (\*)

; mov mq,r2 / calculate offset in i-node for pointer to proper

; / indirect block

; bic $!16,r2

and bl, 0FEh ; bh = 0

push bx ; i-node pointer offset in indirect block (\*)

; 01/03/2013 Max. possible BX (offset) value is 127 (65535/512)

; for this file system (offset 128 to 255 not in use)

; There is always 1 indirect block for this file system

mov ax, word ptr [i.dskp] ; i.dskp[0]

; mov i.dskp(r2),r1

or ax, ax ; R1

jnz short mget\_6 ; 2f

; bne 2f / if no indirect block exists

call alloc

; jsr r0,alloc / allocate a new block

; mov word ptr i.dskp[BX], ax ; R1, block number

mov word ptr [i.dskp], ax ; 03/03/2013

; mov r1,i.dskp(r2) / put block number of new block in i-node

call setimod

; jsr r0,setimod / set i-node modified byte

; AX = new block number

call clear

; jsr r0,clear / clear new block

mget\_6: ;2

; 05/03/2013

; AX = r1, physical block number (of indirect block)

call dskrd ; read indirect block

; jsr r0,dskrd / read in indirect block

pop dx ; R2, get offset (\*)

; mov (sp)+,r2 / get offset

; AX = r1, physical block number (of indirect block)

push ax ; \*\* ; 24/03/2013

; mov r1,-(sp) / save block number of indirect block on stack

; BX (r5) = pointer to buffer (indirect block)

add bx, dx ; / r5 points to first word in indirect block, r2

; add r5,r2 / r5 points to first word in indirect block, r2

; / points to location of inter

mov ax, word ptr [BX] ; put physical block no of block

; in file sought in R1 (AX)

; mov (r2),r1 / put physical block no of block in file

; / sought in r1

or ax, ax

jnz short mget\_7 ; 2f

; bne 2f / if no block exists

call alloc

; jsr r0,alloc / allocate a new block

mov word ptr [BX], ax ; R1

; mov r1,(r2) / put new block number into proper location in

; / indirect block

pop dx ; \*\* ; 24/03/2013

; mov (sp)+,r1 / get block number of indirect block

push dx ; \*\* ; 31/07/2013

push ax ; \* ; 24/03/2013, 31/07/2013 (new block number)

mov ax, dx ; 24/03/2013

; mov (r2),-(sp) / save block number of new block

; AX (r1) = physical block number (of indirect block)

call wslot

; jsr r0,wslot

; AX (r1) = physical block number

; BX (r5) = pointer to buffer (indirect block)

call dskwr

; AX = r1 = physical block number (of indirect block)

; jsr r0,dskwr / write newly modified indirect block

; / back out on disk

pop ax ; \* ; 31/07/2013

; mov (sp),r1 / restore block number of new block

; AX (r1) = physical block number of new block

call clear

; jsr r0,clear / clear new block

mget\_7: ; 2

pop dx ; \*\*

; tst (sp)+ / bump stack pointer

; AX (r1) = Block number of new block

retn

; rts r0

alloc:

; 01/08/2013

; 21/07/2013

; 02/04/2013

; 01/04/2013

;

; get a free block and

; set the corresponding bit in the free storage map

;

; INPUTS ->

; cdev (current device)

; r2

; r3

; OUTPUTS ->

; r1 (physical block number of block assigned)

; smod, mmod, systm (super block), mount (mountable super block)

;

; ((AX = R1)) output

; (Retro UNIX Prototype : 14/11/2012 - 21/07/2012, UNIXCOPY.ASM)

; ((Modified registers: DX, CX))

;mov r2,-(sp) / save r2, r3 on stack

;mov r3,-(sp)

;push cx

push bx ; R2

;push dx ; R3

;mov bx, offset systm ; SuperBlock

mov bx, offset s ; 21/07/2013

; mov $systm,r2 / start of inode and free storage map for drum

cmp byte ptr [cdev], 0

; tst cdev

jna short alloc\_1

; beq 1f / drum is device

mov bx, offset mount

; mov $mount,r2 / disk or tape is device, start of inode and

; / free storage map

alloc\_1: ; 1

mov ax, word ptr [BX]

; mov (r2)+,r1 / first word contains number of bytes in free

; / storage map

shl ax, 1

; asl r1 / multiply r1 by eight gives

; number of blocks in device

shl ax, 1

; asl r1

shl ax, 1

; asl r1

mov cx, ax

;; push cx ;; 01/08/2013

; mov r1,-(sp) / save # of blocks in device on stack

xor ax, ax ; 0

; clr r1 / r1 contains bit count of free storage map

alloc\_2: ; 1

inc bx ; 18/8/2012

inc bx ;

mov dx, word ptr [BX]

; mov (r2)+,r3 / word of free storage map in r3

or dx, dx

jnz short alloc\_3 ; 1f

; bne 1f / branch if any free blocks in this word

add ax, 16

; add $16.,r1

cmp ax, cx

; cmp r1 ,(sp) / have we examined all free storage bytes

jb short alloc\_2

; blo 1b

jmp panic

; jmp panic / found no free storage

alloc\_3: ; 1

shr dx, 1

; asr r3 / find a free block

jc short alloc\_4 ; 1f

; bcs 1f / branch when free block found; bit for block k

; / is in byte k/8 / in bit k (mod 8)

inc ax

; inc r1 / increment bit count in bit k (mod8)

jmp short alloc\_3

; br 1b

alloc\_4: ; 1:

;; pop cx ;; 01/08/2013

; tst (sp)+ / bump sp

; 02/04/2013

call free3

; jsr r0,3f / have found a free block

; 21/8/2012

not dx ; masking bit is '0' and others are '1'

and word ptr [BX], dx ;; 0 -> allocated

; bic r3,(r2) / set bit for this block

; / i.e. assign block

; br 2f

jmp short alloc\_5

free:

; 01/08/2013

; 21/07/2013

; 07/04/2013

;

; calculates byte address and bit position for given block number

; then sets the corresponding bit in the free storage map

;

; INPUTS ->

; r1 - block number for a block structured device

; cdev - current device

; OUTPUTS ->

; free storage map is updated

; smod is incremented if cdev is root device (fixed disk)

; mmod is incremented if cdev is a removable disk

;

; (Retro UNIX Prototype : 01/12/2012, UNIXCOPY.ASM)

; ((Modified registers: DX, CX))

;mov r2,-(sp) / save r2, r3

;mov r3,-(sp)

;push cx

push bx ; R2

;push dx ; R3

call free3

; jsr r0,3f / set up bit mask and word no.

; / in free storage map for block

or word ptr [BX], dx

; bis r3, (r2) / set free storage block bit;

; / indicates free block

; 0 -> allocated, 1 -> free

alloc\_5:

; 07/04/2013

free\_1: ; 2:

; pop dx

; mov (sp)+,r3 / restore r2, r3

pop bx

; mov (sp)+,r2

; pop cx

cmp byte ptr [cdev], 0

; tst cdev / cdev = 0, block structured, drum;

; / cdev = 1, mountable device

ja short alloc\_6 ; 1f

; bne 1f

;mov byte ptr [smod], 1

inc byte ptr [smod]

; incb smod / set super block modified for drum

; AX (r1) = block number

retn

; rts r0

free\_2:

alloc\_6: ; 1:

;mov byte ptr [mmod], 1

inc byte ptr [mmod]

; incb mmod

; / set super block modified for mountable device

; AX (r1) = block number

retn

; rts r0

free3:

; 01/08/2013

; 02/04/2013

;

; free3 is called from 'alloc' and 'free' procedures

;

alloc\_free\_3: ; 3

mov dx, 1

mov cx, ax

; mov r1,r2 / block number, k, = 1

and cx, 0Fh ; 0Fh <-- (k) mod 16

; bic $!7,r2 / clear all bits but 0,1,2; r2 = (k) mod (8)

jz short @f

; bisb 2f(r2),r3 / use mask to set bit in r3 corresponding to

; / (k) mod 8

shl dx, cl

@@:

mov bx, ax

; mov r1,r2 / divide block number by 16

shr bx, 1

; asr r2

shr bx, 1

; asr r2

shr bx, 1

; asr r2

shr bx, 1

; asr r2

; bcc 1f / branch if bit 3 in r1 was 0 i.e.,

; / bit for block is in lower half of word

; swab r3 / swap bytes in r3; bit in upper half of word in free

; / storage map

alloc\_free\_4: ; 1

shl bx, 1 ; 21/8/2012

; asl r2 / multiply block number by 2; r2 = k/8

;add bx, offset systm+2 ; SuperBlock+2

add bx, offset s + 2 ; 21/07/2013

; add $systm+2,r2 / address of word of free storage map for drum

; / with block bit in it

cmp byte ptr [cdev], 0

; tst cdev

jna short alloc\_free\_5

; beq 1f / cdev = 0 indicates device is drum

;add bx, offset mount - offset systm

add bx, offset sb1 - offset sb0 ; 21/07/2013

; add $mount-systm,r2 / address of word of free storage map for

; / mountable device with bit of block to be

; / freed

alloc\_free\_5: ; 1

retn

; rts r0 / return to 'free'

; 2

; .byte 1,2,4,10,20,40,100,200 / masks for bits 0,...,7

iget:

; 07/08/2013

; 31/07/2013

; 28/07/2013

; 18/07/2013

; 17/07/2013

; 09/07/2013 (cdev,mdev)

; 26/04/2013 (mdev)

; 07/04/2013

;

; get a new i-node whose i-number in r1 and whose device is in cdev

; ('iget' returns current i-number in r1, if input value of r1 is 0)

;

; INPUTS ->

; ii - current i-number, rootdir

; cdev - new i-node device

; idev - current i-node device

; imod - current i-node modified flag

; mnti - cross device file i-number

; r1 - i-numbe rof new i-node

; mntd - mountable device number

;

; OUTPUTS ->

; cdev, idev, imod, ii, r1

;

; ((AX = R1)) input/output

;

; (Retro UNIX Prototype : 14/07/2012 - 18/11/2012, UNIXCOPY.ASM)

; ((Modified registers: DX, CX, BX, SI, DI, BP))

mov dl, byte ptr [cdev] ; 18/07/2013

mov dh, byte ptr [idev] ; 07/08/2013

;

cmp ax, word ptr [ii]

; cmp r1,ii / r1 = i-number of current file

jne short iget\_1

; bne 1f

cmp dl, dh

; cmp idev,cdev

; / is device number of i-node = current device

je short @f

; beq 2f

iget\_1: ; 1:

xor bl, bl

cmp byte ptr [imod], bl ; 0

; tstb imod / has i-node of current file

; / been modified i.e., imod set

jna short iget\_2

; beq 1f

mov byte ptr [imod], bl ; 0

; clrb imod / if it has,

; / we must write the new i-node out on disk

push ax

; mov r1,-(sp)

;mov dl, byte ptr [cdev]

push dx

; mov cdev,-(sp)

mov ax, word ptr [ii]

; mov ii,r1

;mov dh, byte ptr [idev]

mov byte ptr [cdev], dh

; mov idev,cdev

inc bl ; 1

; 31/07/2013

mov byte ptr [rw], bl ; 1 == write

;;28/07/2013 rw -> u.rw

;;mov byte ptr [u.rw], bl ; 1 == write

call icalc

; jsr r0,icalc; 1

pop dx

mov byte ptr [cdev], dl

; mov (sp)+,cdev

pop ax

; mov (sp)+,r1

iget\_2: ; 1:

and ax, ax

; tst r1 / is new i-number non zero

jz short iget\_4 ; 2f

; beq 2f / branch if r1=0

; mov dl, byte ptr [cdev]

or dl, dl

; tst cdev / is the current device number non zero

; / (i.e., device =/ drum)

jnz short iget\_3 ; 1f

; bne 1f / branch 1f cdev =/ 0 ;; (cdev != 0)

cmp ax, word ptr [mnti]

; cmp r1,mnti / mnti is the i-number of the cross device

; / file (root directory of mounted device)

jne short iget\_3 ; 1f

; bne 1f

;mov bl, byte ptr [mntd]

inc dl ; move dl, 1 ; 17/07/2013

mov byte ptr [cdev], dl ; 17/07/2013 - 09/07/2013

; mov mntd,cdev / make mounted device the current device

mov ax, word ptr [rootdir]

; mov rootdir,r1

iget\_3: ; 1:

mov word ptr [ii], ax

; mov r1,ii

mov byte ptr [idev], dl ; cdev

; mov cdev,idev

xor bl, bl

; 31/07/2013

mov byte ptr [rw], bl ; 0 == read

;;28/07/2013 rw -> u.rw

;;mov byte ptr [u.rw], bl ; 0 = read

call icalc

; jsr r0,icalc; 0 / read in i-node ii

iget\_4: ; 2:

mov ax, word ptr [ii]

; mov ii,r1

@@:

retn

; rts r0

icalc:

; 31/07/2013

; 28/07/2013

; 17/07/2013

; 07/04/2013

;

; calculate physical block number from i-number then

; read or write that block

;

; 'icalc' is called from 'iget'

;

; for original unix v1:

; / i-node i is located in block (i+31.)/16. and begins 32.\*

; / (i+31)mod16 bytes from its start

;

; for retro unix 8086 v1:

; i-node is located in block (i+47)/16 and

; begins 32\*(i+47) mod 16 bytes from its start

;

; INPUTS ->

; r1 - i-number of i-node

; OUTPUTS ->

; inode r/w

;

; ((AX = R1)) input

;

; (Retro UNIX Prototype : 14/07/2012 - 18/11/2012, UNIXCOPY.ASM)

; ((Modified registers: AX, DX, CX, BX, SI, DI, BP))

;

add ax, 47 ; add 47 to inode number

; add $31.,r1 / add 31. to i-number

push ax

; mov r1,-(sp) / save i+31. on stack

shr ax, 1

; asr r1 / divide by 16.

shr ax, 1

; asr r1

shr ax, 1

; asr r1

shr ax, 1

; asr r1 / r1 contains block number of block

; / in which i-node exists

call dskrd

; jsr r0,dskrd / read in block containing i-node i.

; 31/07/2013

cmp byte ptr [rw], 0 ; Retro Unix 8086 v1 feature !

;; 28/07/2013 rw -> u.rw

;;cmp byte ptr [u.rw], 0 ; Retro Unix 8086 v1 feature !

; tst (r0)

jna short icalc\_1

; beq 1f / branch to wslot when argument

; / in icalc call = 1

; AX = r1 = block number

call wslot

; jsr r0,wslot / set up data buffer for write

; / (will be same buffer as dskrd got)

; BX = r5 points to first word in data area for this block

icalc\_1: ; 1:

pop dx

and dx, 0Fh ; (i+47) mod 16

; bic $!17,(sp) / zero all but last 4 bits;

; / gives (i+31.) mod 16

shl dx, 1

shl dx, 1

shl dx, 1

shl dx, 1

shl dx, 1

; DX = 32 \* ((i+47) mod 16)

mov si, bx ; bx points 1st word of the buffer

add si, dx ; dx is inode offset in the buffer

; SI (r5) points to first word in i-node i.

; mov (sp)+,mq / calculate offset in data buffer;

; / 32.\*(i+31.)mod16

; mov $5,lsh / for i-node i.

; add mq,r5 / r5 points to first word in i-node i.

;mov di, offset inode

mov di, offset i ; 17/07/2013

; mov $inode,r1 / inode is address of first word

; / of current i-node

mov cx, 16 ; CX = r3

; mov $16.,r3

; 31/07/2013

cmp byte ptr [rw], ch ; 0 ;; Retro Unix 8086 v1 feature !

;;28/07/2013 rw -> u.rw

;;cmp byte ptr [u.rw], ch ; 0 ;; Retro Unix 8086 v1 feature !

; tst (r0)+ / branch to 2f when argument in icalc call = 0

jna short icalc\_3

; beq 2f / r0 now contains proper return address

; / for rts r0

icalc\_2: ; 1:

xchg si, di

; over write old i-node (in buffer to be written)

rep movsw

; mov (r1)+,(r5)+ / over write old i-node

; dec r3

; bgt 1b

call dskwr

; jsr r0,dskwr / write inode out on device

retn

; rts r0

icalc\_3: ; 2:

; copy new i-node into inode area of (core) memory

rep movsw

; mov (r5)+,(r1)+ / read new i-node into

; / "inode" area of core

; dec r3

; bgt 2b

retn

; rts r0

access:

; 29/04/2013 (AX register preserved)

; 24/04/2013

; check whether user is owner of file or user has read or write

; permission (based on i.flgs).

;

; INPUTS ->

; r1 - i-number of file

; u.uid

; arg0 -> (owner flag mask)

; Retro UNIX 8086 v1 feature -> owner flag mask in DL (DX)

; OUTPUTS ->

; inode (or jump to error)

; ((AX = R1)) input/output

; ((Modified registers: CX, BX, SI, DI, BP))

;

push dx ; flags

call iget

; jsr r0,iget / read in i-node for current directory

; / (i-number passed in r1)

mov cx, word ptr [i.flgs]

; mov i.flgs,r2

pop dx

mov dh, byte ptr [u.uid\_] ; 29/04/2013 al -> dh

cmp dh, byte ptr [i.uid] ; 29/04/2013

; cmpb i.uid,u.uid / is user same as owner of file

jne short access\_1

; bne 1f / no, then branch

shr cl, 1

; asrb r2 / shift owner read write bits into non owner

; / read/write bits

shr cl, 1

; asrb r2

access\_1: ; 1:

and cl, dl

; bit r2,(r0)+ / test read-write flags against argument

; / in access call

jnz short access\_2

; bne 1f

or dh, dh ; 29/04/2013 al -> dh

; tstb u.uid

jnz error

; beq 1f

; jmp error

access\_2: ; 1:

retn

; rts r0

setimod:

; 31/07/2013

; 09/04/2013

; 'setimod' sets byte at location 'imod' to 1; thus indicating that

; the inode has been modified. Also puts the time of modification

; into the inode.

;

; (Retro UNIX Prototype : 14/07/2012 - 23/02/2013, UNIXCOPY.ASM)

; ((Modified registers: DX, CX, BX))

; push dx

push ax

mov byte ptr [imod], 1

; movb $1,imod / set current i-node modified bytes

; Erdogan Tan, 14-7-2012

call epoch

; mov s.time,i.mtim

; / put present time into file modified time

; mov s.time+2,i.mtim+2

mov word ptr [i.mtim], ax

mov word ptr [i.mtim]+2, dx

; Retro UNIX 8086 v1 modification !

mov cx, word ptr [i.ctim]

mov bx, word ptr [i.ctim]+2

test cx, bx

jnz short @f

mov word ptr [i.ctim], ax

mov word ptr [i.ctim]+2, dx

@@: ; 31/07/2013

pop ax

;pop dx

retn

; rts r0

itrunc:

; 01/08/2013

; 23/04/2013

; 'itrunc' truncates a file whose i-number is given in r1

; to zero length.

;

; INPUTS ->

; r1 - i-number of i-node

; i.dskp - pointer to contents or indirect block in an i-node

; i.flgs - large file flag

; i.size - size of file

; OUTPUTS ->

; i.flgs - large file flag is cleared

; i.size - set to 0

; i.dskp .. i.dskp+16 - entire list is cleared

; setimod - set to indicate i-node has been modified

; r1 - i-number of i-node

;

; ((AX = R1)) input/output

;

; (Retro UNIX Prototype : 01/12/2012 - 10/03/2013, UNIXCOPY.ASM)

; ((Modified registers: DX, CX, BX, SI, DI, BP))

call iget

; jsr r0,iget

mov si, offset i.dskp

; mov $i.dskp,r2 / address of block pointers in r2

itrunc\_1: ; 1:

lodsw

; mov (r2)+,r1 / move physical block number into r1

or ax, ax

jz short itrunc\_5

; beq 5f

push si

; mov r2,-(sp)

test word ptr [i.flgs], 1000h

; bit $10000,i.flgs / test large file bit?

jz short itrunc\_4

; beq 4f / if clear, branch

push ax

; mov r1,-(sp) / save block number of indirect block

call dskrd

; jsr r0,dskrd / read in block, 1st data word

; / pointed to by r5

; BX = r5 = Buffer data address (the 1st word)

mov cx, 256

; mov $256.,r3 / move word count into r3

mov si, bx

itrunc\_2: ; 2:

lodsw

; mov (r5)+,r1 / put 1st data word in r1;

; / physical block number

and ax, ax

jz short itrunc\_3

; beq 3f / branch if zero

push cx

; mov r3,-(sp) / save r3, r5 on stack

;push si

; mov r5,-(sp)

call free

; jsr r0,free / free block in free storage map

;pop si

; mov(sp)+,r5

pop cx

; mov (sp)+,r3

itrunc\_3: ; 3:

loop itrunc\_2

; dec r3 / decrement word count

; bgt 2b / branch if positive

pop ax

; mov (sp)+,r1 / put physical block number of

; / indirect block

; 01/08/2013

and word ptr [i.flgs], 0EFFFh ; 1110111111111111b

itrunc\_4: ; 4:

call free

; jsr r0,free / free indirect block

pop si

; mov (sp)+,r2

itrunc\_5: ; 5:

cmp si, offset i.dskp+16

; cmp r2,$i.dskp+16.

jb short itrunc\_1

; bne 1b / branch until all i.dskp entries check

; 01/08/2013

;and word ptr [i.flgs], 0EFFFh ; 1110111111111111b

; bic $10000,i.flgs / clear large file bit

mov di, offset i.dskp

mov cx, 8

xor ax, ax

mov word ptr [i.size\_], ax ; 0

; clr i.size / zero file size

rep stosw

; jsr r0,copyz; i.dskp; i.dskp+16.

; / zero block pointers

call setimod

; jsr r0,setimod / set i-node modified flag

mov ax, word ptr [ii]

; mov ii,r1

retn

; rts r0

imap:

; 26/04/2013

; 'imap' finds the byte in core (superblock) containing

; allocation bit for an i-node whose number in r1.

;

; INPUTS ->

; r1 - contains an i-number

; fsp - start of table containing open files

; OUTPUTS ->

; r2 - byte address of byte with the allocation bit

; mq - a mask to locate the bit position.

; (a 1 is in calculated bit posisiton)

;

; ((AX = R1)) input/output

; ((DL/DX = MQ)) output

; ((BX = R2)) output

;

; (Retro UNIX Prototype : 02/12/2012, UNIXCOPY.ASM)

; ((Modified registers: DX, CX, BX, SI))

;

; / get the byte that has the allocation bit for

; / the i-number contained in r1

;mov dx, 1

mov dl, 1

; mov $1,mq / put 1 in the mq

mov bx, ax

; mov r1,r2 / r2 now has i-number whose byte

; / in the map we must find

sub bx, 41

; sub $41.,r2 / r2 has i-41

mov cl, bl

; mov r2,r3 / r3 has i-41

and cl, 7

; bic $!7,r3 / r3 has (i-41) mod 8 to get

; / the bit position

jz short @f

;shl dx, cl

shl dl, cl

; mov r3,lsh / move the 1 over (i-41) mod 8 positions

@@: ; / to the left to mask the correct bit

shr bx, 1

; asr r2

shr bx, 1

; asr r2

shr bx, 1

; asr r2 / r2 has (i-41) base 8 of the byte number

; / from the start of the map

; mov r2,-(sp) / put (i-41) base 8 on the stack

;mov si, offset systm

mov si, offset s ; 21/07/2013

; mov $systm,r2 / r2 points to the in-core image of

; / the super block for drum

;cmp word ptr [cdev], 0

cmp byte ptr [cdev], 0

; tst cdev / is the device the disk

jna short @f

; beq 1f / yes

;add si, offset mount - offset systm

add si, offset mount - offset s ; 21/07/2013

; add $mount-systm,r2 / for mounted device,

; / r2 points to 1st word of its super block

@@: ; 1:

add bx, word ptr [SI] ;; add free map size to si

; add (r2)+,(sp) / get byte address of allocation bit

add bx, si

; add (sp)+,r2 / ?

add bx, 4 ;; inode map offset in superblock

;; (2 + free map size + 2)

; add $2,r2 / ?

; DL/DX (MQ) has a 1 in the calculated bit position

; BX (R2) has byte address of the byte with allocation bit

retn

; rts r0