Description of PC/XT Emulator for AMIGA 2000

AMIGA ACCESS: Amiga Interface Offset Address = Base Addr.

Base Addr. + (00000 – 1FFFF) : Byte Access
Base Addr. + (20000 – 3FFFF) : Word Access
Base Addr. + (40000 – 5FFFF) : Graphic Access
Base Addr. + (60000 – 7FFFF) : I/O Register Access

INTERFACE MEMORY MAP:

Interface Offset Address	Size	Usage
00000 . . . 0FFFF	64K	DISK BUFFER RAM
10000 . . . 17FFF	32K	COLOR VIDEO RAM
18000 . . . 1BFFF	16K	PARAMETER RAM
1C000 . . . 1DFFF	8K	MONO VIDEO RAM
1E000 . . . 1FFFF	8K	IO-PAGE

Kinds of memory access on the following pages:
B = Byte access
G = Graphic access
W = Word access

(*) selectable by BIT 5 and 6 of the MODE REGISTER

BIT 5 = SEL1
BIT 6 = SEL2

PC MEMORY AND I/O MAP:

PC Address	Range	Size	Usage	kind of access	Amiga Interface Offset Address
0000 ...	03FF	1K	IO-PAGE	B	1E000 ... 1FFFF
				W	3E000 ... 3FFFF
				G	5E000 ... 5FFFF
					7E000 ... 7FFFF
A0000 ...	AFFFF	64K	DISK BUFFER RAM (*)	B	00000 ... 0FFFF
			(*)	W	20000 ... 2FFFF
			(*)	G	40000 ... 4FFFF
B0000 ...	B1FFF	8K	MONO VIDEO RAM	B	1C000 ... 1DFFF
				W	3C000 ... 3DFFF
				G	5C000 ... 5DFFF
B8000 ...	BFFFF	32K	COLOR VIDEO RAM	B	10000 ... 17FFF
				W	30000 ... 37FFF
				G	50000 ... 57FFF
E0000 ...	EFFFF	64K	DISK BUFFER RAM (*)	B	00000 ... 0FFFF
			(*)	W	20000 ... 2FFFF
			(*)	G	40000 ... 4FFFF
D0000 ...	DFFFF	64K	DISK BUFFER RAM (*)	B	00000 ... 0FFFF
			(*)	W	20000 ... 2FFFF
			(*)	G	40000 ... 4FFFF
F0000 ...	F3FFF	16K	PARAMETER RAM	B	18000 ... 1BFFF
				W	38000 ... 3BFFF
				G	58000 ... 5BFFF

AMIGA MEMORY MAP:

Amiga Interface Offset Address	PC Address Range	Size	Usage	kind of access
00000 ... 0FFFF	A0000 ... AFFFF	64K	DISK BUFFER RAM (*)	B
00000 ... 0FFFF	D0000 ... DFFFF	64K	DISK BUFFER RAM (*)	B
00000 ... 0FFFF	E0000 ... EFFFF	64K	DISK BUFFER RAM (*)	B
10000 ... 17FFF	B8000 ... BFFFF	32K	COLOR VIDEO RAM	B
18000 ... 1BFFF	F0000 ... F3FFF	16K	PARAMETER RAM	B
1C000 ... 1DFFF	B0000 ... B1FFF	8K	MONO VIDEO RAM	B
1E000 ... 1FFFF	0000 ... 03FF	1K	IO-PAGE	B
20000 ... 2FFFF	A0000 ... AFFFF	64K	DISK BUFFER RAM (*)	W
20000 ... 2FFFF	D0000 ... DFFFF	64K	DISK BUFFER RAM (*)	W
20000 ... 2FFFF	E0000 ... EFFFF	64K	DISK BUFFER RAM (*)	W
30000 ... 37FFF	B8000 ... BFFFF	32K	COLOR VIDEO RAM	W
38000 ... 3BFFF	F0000 ... F3FFF	16K	PARAMETER RAM	W
3C000 ... 3DFFF	B0000 ... B1FFF	8K	MONO VIDEO RAM	W
3E000 ... 3FFFF	0000 ... 03FF	1K	IO-PAGE	W
40000 ... 4FFFF	A0000 ... AFFFF	64K	DISK BUFFER RAM (*)	G

40000 . . . 4FFFF	D0000 . . . DFFFF	64K	DISK BUFFER RAM (*)	G
40000 . . . 4FFFF	E0000 . . . EFFFF	64K	DISK BUFFER RAM (*)	G
50000 . . . 57FFF	B8000 . . . BFFFF	32K	COLOR VIDEO RAM	G
58000 . . . 5BFFF	F0000 . . . F3FFF	16K	PARAMETER RAM	G
5C000 . . . 5DFFF	B0000 . . . B1FFF	8K	MONO VIDEO RAM	G
5E000 . . . 5FFFF	0000 . . . 03FF	1K	IO-PAGE	G
7E000 . . . 7FFFF	0000 . . . 03FF	1K	IO-PAGE	

AT MEMORY and I/O MAP:

PC Address	Range	Size	Usage	kind of access	Amiga Interface Offset Address
0000 . . .	03FF	1K	IO-PAGE	B	1E000 . . . 1FFFF
				W	3E000 . . . 3FFFF
				G	5E000 . . . 5FFFF
					7E000 . . . 7FFFF
A0000 . . .	AFFFF	64K	DISK BUFFER RAM (*)	B	00000 . . . 0FFFF
			(*)	W	20000 . . . 2FFFF
			(*)	G	40000 . . . 4FFFF
B0000	B1FFF	8K	MONO VIDEO RAM	B	1C000 . . . 1DFFF
				W	3C000 . . . 3DFFF
				G	5C000 . . . 5DFFF
B8000 . . .	BFFFF	32K	COLOR VIDEO RAM	B	10000 . . . 17FFF
				W	30000 . . . 37FFF
				G	50000 . . . 57FFF
D0000 . . .	D3FFF	16K	PARAMETER RAM	B	18000 . . . 1BFFF
				W	38000 . . . 3BFFF
				G	58000 . . . 5BFFF
D4000 . . .	DFFFF	64K	DISK BUFFER RAM (*)	B	04000 . . . 0FFFF
			(*)	W	24000 . . . 2FFFF
			(*)	G	44000 . . . 4FFFF

AMIGA MEMORY MAP:

Amiga Interface Offset Address	AT Address Range	Size	Usage	kind of access
00000 . . . 0FFFF	A0000 . . . AFFFF	64K	DISK BUFFER RAM (*)	B
00000 . . . 03FFF	CAN NOT BE ACCESSED BY THE AT			
04000 . . . 0FFFF	D4000 . . . DFFFF	48K	DISK BUFFER RAM (*)	B
10000 . . . 17FFF	B8000 . . . BFFFF	32K	COLOR VIDEO RAM	B
18000 . . . 1BFFF	D0000 . . . D3FFF	16K	PARAMETER RAM	B
1C000 . . . 1DFFF	B0000 . . . B1FFF	8K	MONO VIDEO RAM	B
1E000 . . . 1FFFF	0000 . . . 03FF	1K	IO-PAGE	B
20000 . . . 2FFFF	A0000 . . . AFFFF	64K	DISK BUFFER RAM (*)	W
20000 . . . 23FFF	CAN NOT BE ACCESSED BY THE AT			

AMIGA MEMORY MAP:

Amiga Interface Offset Address	AT Address Range	Size	Usage	kind of access
24000 ... 2FFFF	D4000 ... DFFFF	48K	DISK BUFFER RAM (*)	W
30000 ... 37FFF	B8000 ... BFFFF	32K	COLOR VIDEO RAM	W
38000 ... 3BFFF	D0000 ... D3FFF	16K	PARAMETER RAM	W
3C000 ... 3DFFF	B0000 ... B1FFF	8K	MONO VIDEO RAM	W
3E000 ... 3FFFF	0000 ... 03FF	1K	IO-PAGE	W
40000 ... 4FFFF	A0000 ... AFFFF	64K	DISK BUFFER RAM (*)	G
40000 ... 43FFF	CAN NOT BE ACCESSED BY THE AT			
44000 ... 4FFFF	D4000 ... DFFFF	48K	DISK BUFFER (*)	G
50000 ... 57FFF	B8000 ... BFFFF	32K	COLOR VIDEO RAM	G
58000 ... 5BFFF	D0000 ... D3FFF	16K	PARAMETER RAM	G
5C000 ... 5DFFF	B0000 ... B1FFF	8K	MONO VIDEO RAM	G
5E000 ... 5FFFF	0000 ... 03FF	1K	IO-PAGE	G
7E000 ... 7FFFF	0000 ... 03FF	1K	IO-PAGE	

PC/AT I/O REGISTER MAP

PC/AT I/O Address	Usage				Offset Address INTERFACE / AMIGA	
60	KEYBOARD DATA			(W)	1E41F	7E41F
61	SYSTEM REGISTER			(W)	1E05F	7E05F
62	SYSTEM STATUS			(W)	1E03F	7E03F
2F8	COM2	TRANSMIT DATA	(DLAB=0)	(W)	1E07D	7E07D
2F8	"	RECEIVE DATA	(DLAB=0)	(R)	1E09D	7E09D
2F8	"	RESET IRQ3_b	(DLAB=0)	(R)	1E09D	7E09D
2F9	"	INTERRUPT CONTROL	(DLAB=0)	(W)	1E0BD	7E0BD
2F9	"	INTERRUPT CONTROL	(DLAB=0)	(R)	1E0DD	7E0DD
2F8	"	DIVISOR LATCH (LSB)	(DLAB=1)	(R/W)	1E07F	7E07F
2F8	"	RESET IRQ3_b	(DLAB=1)	(R)	1E07F	7E07F
2F9	"	DIVISOR LATCH (MSB)	(DLAB=1)	(R/W)	1E09F	7E09F
2FA	COM2	INTERRUPT ACKN		(R)	1E0FF	7E0FF
2FA		DUMMY		(W)	1E01F	7E01F
2FB	"	LINE CONTROL	(DLAB=BIT 7)	(W)	1E11F	7E11F
2FB		DUMMY		(R)	1E01F	7E01F
2FC	"	MODEM CONTROL		(W)	1E13F	7E13F
2FC		DUMMY		(R)	1E01F	7E01F
2FD	"	LINE STATUS		(R)	1E15F	7E15F
2FD		DUMMY		(W)	1E01F	7E01F
2FE	"	MODEM STATUS		(R)	1E17F	7E17F
2FE		DUMMY		(W)	1E01F	7E01F
2FF		DUMMY		(R/W)	1E01F	7E01F
378	LPT1	PRINTER DATA		(R/W)	1E19F	7E19F
379	"	STATUS		(R)	1E1BF	7E1BF

PC/AT I/O Address	Usage			Offset Address INTERFACE / Amiga	
379	"	RESET IRQ7	(R)	1E1BF	7E1BF
379	"	INTERRUPT CONTROL	(W)	1E19F	7E19F
		BIT 6 = 0 : ON			
		BIT 6 = 0 : OFF			
37A	"	CONTROL	(W)	1E1DF	7E1DF
37A	"	CONTROL	(R)	1E19F	7E19F
3B0	MONO	CRT ADDRESS INDEX REGISTER	(W)	1E1FF	7E1FF
3B0	"	RESET IRQ3_a	(R)	1E01F	7E01F
3B2	"	CRT ADDRESS INDEX REGISTER	(W)	1E1FF	7E1FF
3B2		DUMMY	(R)	1E01F	7E01F
3B4	"	CRT ADDRESS INDEX REGISTER	(W)	1E1FF	7E1FF
3B4		DUMMY	(R)	1E01F	7E01F
3B6	"	CRT ADDRESS INDEX REGISTER	(W)	1E1FF	7E1FF
3B6		DUMMY	(R)	1E01F	7E01F
3B1	"	CRT DATA REGISTER	(R/W)		s.b.
3B3	"	CRT DATA REGISTER	(R/W)		s.b.
3B5	"	CRT DATA REGISTER	(R/W)		s.b.
3B7	"	CRT DATA REGISTER	(R/W)		s.b
		LAST WRITE ON INDEX = 00		1E2A1	7E2A1
		LAST WRITE ON INDEX = 01		1E2A3	7E2A3
		LAST WRITE ON INDEX = 02		1E2A5	7E2A5
		LAST WRITE ON INDEX = 03		1E2A7	7E2A7
		LAST WRITE ON INDEX = 04		1E2A9	7E2A9
		LAST WRITE ON INDEX = 05		1E2AB	7E2AB
		LAST WRITE ON INDEX = 06		1E2AD	7E2AD
		LAST WRITE ON INDEX = 07		1E2AF	7E2AF
		LAST WRITE ON INDEX = 08		1E2B1	7E2B1
		LAST WRITE ON INDEX = 09		1E2B3	7E2B3
		LAST WRITE ON INDEX = 0A		1E2B5	7E2B5
		LAST WRITE ON INDEX = 0B		1E2B7	7E2B7
		LAST WRITE ON INDEX = 0C		1E2B9	7E2B9
		LAST WRITE ON INDEX = 0D		1E2BB	7E2BB
		LAST WRITE ON INDEX = 0E		1E2BD	7E2BD
		LAST WRITE ON INDEX = 0F		1E2BF	7E2BF
3B8	MONO	CONTROL REGISTER	(W)	1E2FF	7E2FF
3BA	MONO	STATUS REGISTER	(R)	——	——
		BIT 0 : H-SYNC (18KHz)			
		BIT 3 : V-SYNC (50 Hz)			
3BA	DUMMY		(W)	1E01F	7E01F
3BB	DUMMY		(R/W)	1E01F	7E01F
3BC	DUMMY		(R/W)	1E01F	7E01F
3BD	DUMMY		(R/W)	1E01F	7E01F
3BE	DUMMY		(R/W)	1E01F	7E01F
3BF	DUMMY		(R/W)	1E01F	7E01F

PC/AT I/O Address	Usage		Offset Address INTERFACE/		Amiga
3D0	COLOR	CRT ADDRESS INDEX REGISTER	(W)	1E21F	7E21F
3D0		DUMMY	(R)	1E01F	7E01F
3D2	"	CRT ADDRESS INDEX REGISTER	(W)	1E21F	7E21F
3D2		DUMMY	(R)	1E01F	7E01F
3D4	"	CRT ADDRESS INDEX REGISTER	(W)	1E21F	7E21F
3D4		DUMMY	(R)	1E01F	7E01F
3D6	"	CRT ADDRESS INDEX REGISTER	(W)	1E21F	7E21F
3D6		DUMMY	(R)	1E01F	7E01F
3D1	"	CRT DATA REGISTER	(R/W)		s.b.
3D3	"	CRT DATA REGISTER	(R/W)		s.b.
3D5	"	CRT DATA REGISTER	(R/W)		s.b.
3D7	"	CRT DATA REGISTER	(R/W)		s.b
		LAST WRITE ON INDEX = 00		1E2C1	7E2C1
		LAST WRITE ON INDEX = 01		1E2C3	7E2C3
		LAST WRITE ON INDEX = 02		1E2C5	7E2C5
		LAST WRITE ON INDEX = 03		1E2C7	7E2C7
		LAST WRITE ON INDEX = 04		1E2C9	7E2C9
		LAST WRITE ON INDEX = 05		1E2CB	7E2CB
		LAST WRITE ON INDEX = 06		1E2CD	7E2CD
		LAST WRITE ON INDEX = 07		1E2CF	7E2CF
		LAST WRITE ON INDEX = 08		1E2D1	7E2D1
		LAST WRITE ON INDEX = 09		1E2D3	7E2D3
		LAST WRITE ON INDEX = 0A		1E2D5	7E2D5
		LAST WRITE ON INDEX = 0B		1E2D7	7E2D7
		LAST WRITE ON INDEX = 0C		1E2D9	7E2D9
		LAST WRITE ON INDEX = 0D		1E2DB	7E2DB
		LAST WRITE ON INDEX = 0E		1E2DD	7E2DD
		LAST WRITE ON INDEX = 0F		1E2DF	7E2DF
3D8	COLOR	CONTROL REGISTER	(W)	1E23F	7E23F
3D8	DUMMY		(R)	1E01F	7E01F
3D9	COLOR	SELECT REGISTER	(W)	1E25F	7E25F
3D9	DUMMY		(R)	1E01F	7E01F
3DA	COLOR	STATUS REGISTER	(R)	——	——
	BIT 0 :	H-SYNC (18KHz)			
	BIT 3 :	V-SYNC (50 Hz)			
3DA	DUMMY		(W)	1E01F	7E01F
3DD	DISPLAY SYSTEM REGISTER		(W)	1E29F	7E29F
3DD	DUMMY		(R)	1E01F	7E01F
3DE	DUMMY		(R/W)	1E01F	7E01F
3DF	DUMMY		(R/W)	1E01F	7E01F

AMIGA I/O MEMORY MAP (REGISTER DESCRIPTION)

AMIGA	Register	Interface / Memory		Offset Address INTERFACE	/AMIGA
AMIGA	INTERRUPT STATUS	read register		1FF1	7FF1
PC	INTERRUPT STATUS	read register		1FFF3	7FFF3
NEGATE PC RESET		read register		1FFF5	7FFF5
MODE REGISTER		read register /	write register	1FFF7	7FFF7
INTERRUPT MASK		read memory /	write register	1FFF9	7FFF9
PC INTERRUPT CONTROL		read memory /	write register	1FFFB	7FFFB
CONTROL REGISTER		read memory /	write register	1FFFD	7FFFD
KEYBOARD REGISTER		read memory /	write register	1FFFF	7FFFF

PC SIDE

System Status Register:

How to Enable/Disable Interrupts from Amiga to PC

A write access to this register (i/o location 62 hex) forces a /SYSINT interrupt on the AMIGA side

A write access to bit 6 of i/o location 379 hex enables/disables the AMIGA forced interrupts IRQ1 (keyboard), IRQ3 (serial interface COM2) and IRQ7 (parallel interface LPT1) as follows:

D6	Function
0	interrupts enabled
1	interrupts disabled

Note:

The access to i/o location 379 hex is enabled if PARON is high. That is, the AMIGA has to write a "1" to MODE REGISTER bit 1. (See 'Amiga Mode Register.')

The following initialization routine must be used to allow an external printer card on the pc side:

```
AMIGA: set MODE REGISTER bit 1 to "1"  ; switch parallel
                                       ; interface on
PC   : set i/o location 379 hex bit 6 to "0"; keyboard and
                                       ; serial interr. off
AMIGA: set MODE REGISTER bit 1 to "0"  ; switch parallel
                                       ; interface off
```

Now the keyboard and the serial interface emulation is enabled, the parallel interface emulation is disabled.

How to Clear an Asserted Interrupt Signal

INT	Negation
IRQ3_a	Read to i/o location 3b0 hex
IRQ3_b	Read com2 register 2F8 hex
IRQ7	Read line printer status register 379 hex

AMIGA SIDE

All registers on the memory locations 1FFF0 TO 1FFFF are only accessable from the AMIGA side.

Amiga Interrupt Status Register (R) (1FFF1 / 17FFF1)

Reading this register returns the interrupt events caused on PC accesses as follows:

Bit no.	Function	
0	Mono Video Ram	(/MINT)
1	Color Video Ram	(/GINT)
2	Mono CRT	(/CRT1INT)
3	Color CRT	(/CRT2INT)
4	Keyboard Register	(/ENBKB)
5	LPT1 Control Reg	(/LPT1INT)
6	COM2 Data Reg	(/COM2INT)
7	see PC System Status Res	(/SYSINT)

The event was valid if the bit is set to "1". After reading the register all bits turns to "0" automatically and the interrupt flag will be negated.

PC Interrupt Status Register (R) (1FFF3 / 7FFF3)

Reading this register returns the pending PC interrupts on the lower nibble. The PC interrupt is asserted as shown by the corresponding bit in the table.

Bit no.	Function	Asserted if
0	IRQ1 (Keyboard interrupt)	1
1	IRQ3_a	0
2	IRQ3_b	0
3	IRQ7	0
4-7	NOT USED, always HIGH	

Bit 1 and bit 2 (IRQ3_a and IRQ3_b) are externally "ORed" to IRQ3

Negate PC Reset (R) (1FFF5 / 7FFF5)

A read access to this register negates the PC reset line and allows the PC to start the boot procedure. On power-on the PC reset line is asserted (default).

Mode Register (R/W) (1FFF7 / 7FFF7)

Reading this register returns system configuration information

Bit no.	Name	Function
0	SERON	serial interface enabled
1	PARON	parallel interface enabled
2	KEYON	keyboard interface enabled
3	MON	monochrome display emulation enabled
4	COLOR	color display emulation enabled
5	SEL1	select the PC/AT memory bank, s.b.
6	SEL2	select the PC/AT memory bank, s.b.
7	PC/AT	LOW = AT mode HIGH = PC mode

Writing to this register sets system configuration information

Bit no.	Name	Function
0	SERON	switch serial interface on
1	PARON	switch parallel interface on
2	KEYON	switch keyboard interface on
3	MON	enable monochrome display emulation
4	COLOR	enable color display emulation
5	SEL1	PC/AT memory bank select, s.b.
6	SEL2	PC/AT memory bank select, s.b.
7	/STOPCLK	LOW = disable the clock for video retrace and keyboard HIGH = enable the clock for video retrace and keyboard

SEL2	SEL1	PC memory	AT memory
0	0	not used	not used
0	1	A0000 – AFFFF HEX	A0000 – AFFFF HEX
1	0	D0000 – DFFFF HEX	D4000 – DFFFF HEX
1	1	E0000 – EFFFF HEX	not used

Interrupt Mask Register (R/W) (1FFF9 / 7FFF9)

You can mask each PC interrupt event separately by writing a "1" to the corresponding bit as shown below.

Bit no.	Maskable Event (cmp. to Amiga interrupt status reg.)
0	/MINT
1	/GINT
2	/CRT1INT
3	/CRT2INT
4	ENKBKB
5	/LPT1INT
6	/COM2INT
7	/SYSINT

PC Interrupt Control Register (R/W) (1FFFB / 7FFFB)

A PC interrupt can be forced by writing a "0" to the corresponding bit of the lower nibble except the keyboard interrupt, which can be asserted by writing a "1", as shown below:

Bit no.	Asserted PC interrupt level
0	KBSTART (start keyboard shift-register)
1	IRQ3_a (forces interrupt IRQ3)
2	IRQ3_b (forces interrupt IRQ3)
3	IRQ7

Bit 1 and bit 2 (IRQ3_a and IRQ_b) are externally "ORed" to IRQ3

Control Register (R/W) (1FFFD / 7FFFD)

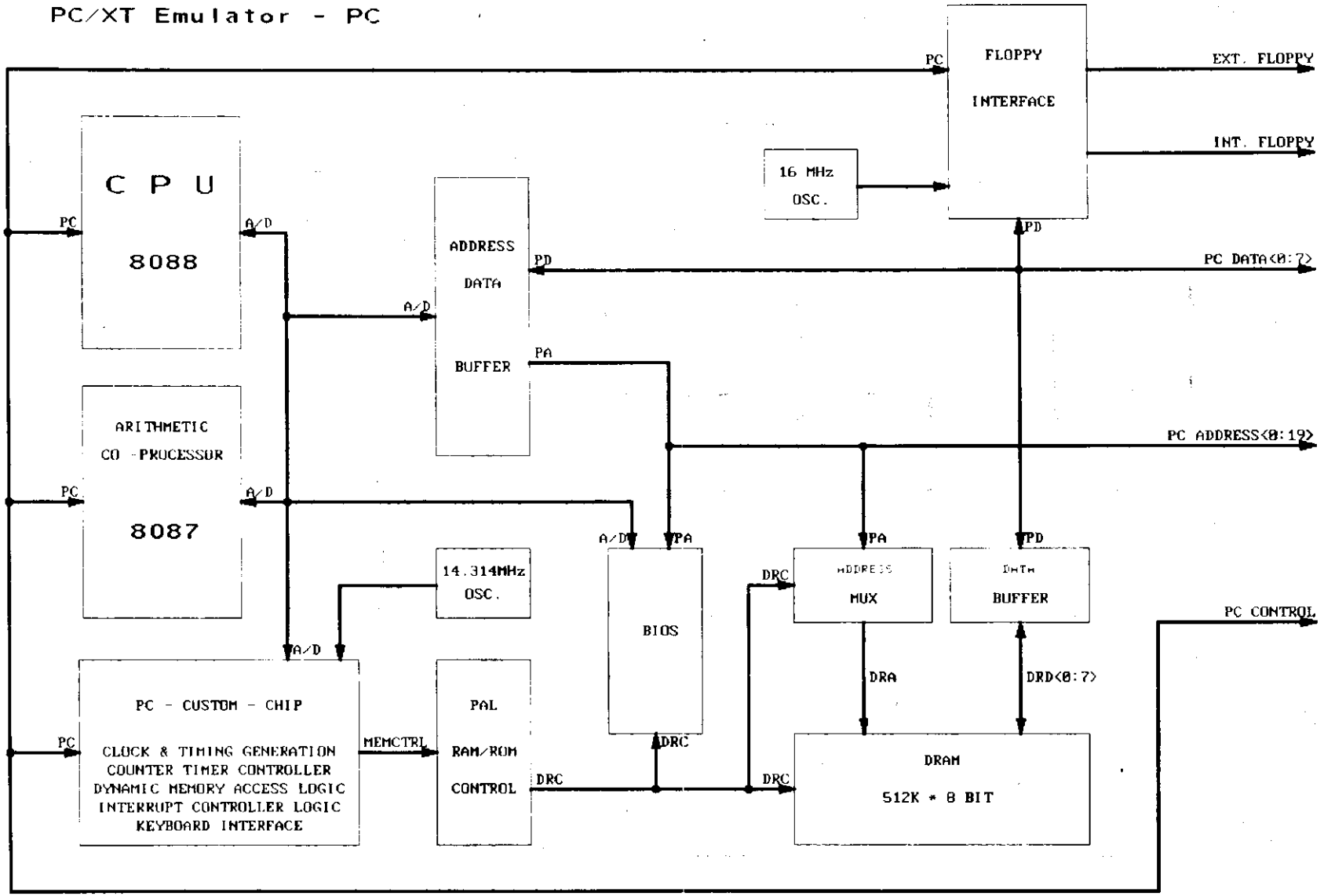
All control function will be done by writing a "0" to the corresponding bit. Only bits 0 to 4 are used.

Bit no.	usage
0	general interrupt enable to the AMIGA
1	general interrupt disable to the AMIGA (default)
2	assert the PC reset line
3	negate all PC interrupt levels except the keyboard interrupt
4	reset line printer BUSY (port 379 hex bit 7) The line printer BUSY bit will be set by writing a "1" to bit 0 of port 37A hex from PC side.

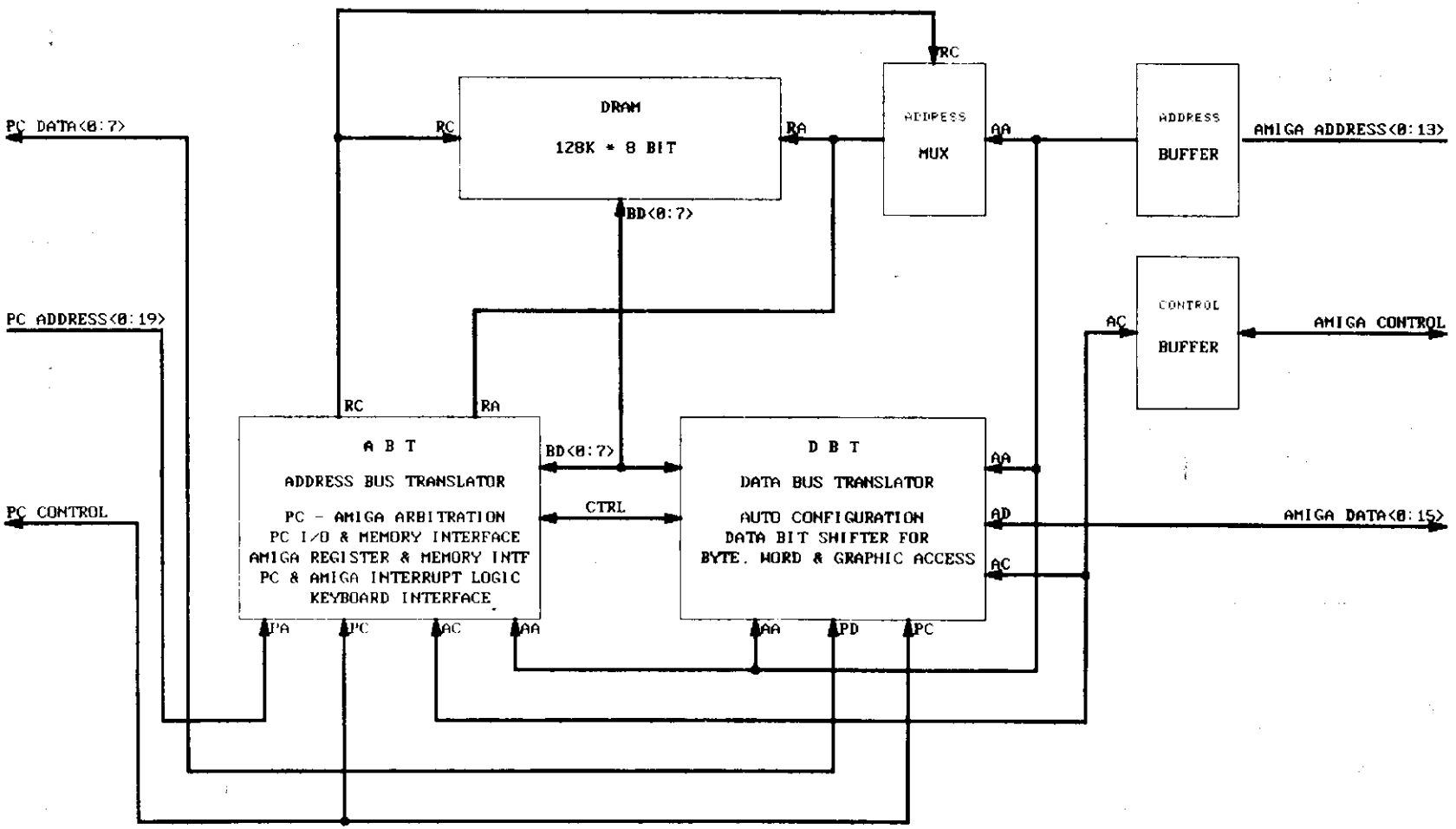
Keyboard Register (R/W) (1FFFF / 7FFFF)

Keyboard emulation is done by writing a character to this register and then asserting a "1" to bit 0 of the PC INTERRUPT CONTROL REGISTER.

PC/XT Emulator - PC



PC/XT Emulator - interface



BIOS Entry Points

VIDEO ENTRY POINT VIA S/W INT 10H

SET VIDEO MODE (AH = 00H)

INPUT: AL = VIDEO MODE (0-7)
- 0: 40 × 25 alpha b/w
- 1: 40 × 25 alpha 16 colors
- 2: 80 × 25 alpha b/w
- 3: 80 × 25 alpha 16 colors
- 4: 320 × 200 graphics 4 colors
- 5: 320 × 200 graphics b/w
- 6: 640 × 200 graphics monochrome
- 7: 80 × 25 alpha monochrome

SET CURSOR TYPE (AH = 01H)

INPUT: CH = START LINE OF CURSOR (BITS 0-4)
CURSOR CONTROL OPERATION (BITS 5-6)
- 00 = NON-BLINK
- 01 = DON'T DISPLAY CURSOR
- 10 = BLINK @ 1/16 FIELD RATE
- 11 = BLINK @ 1/32 FIELD RATE

CL = END LINE OF CURSOR (BITS 0-4)

SET CURSOR POSITION (AH = 02H)

INPUT: BH = Page # if CRT mode is 0 -> 3
(0 if graphics or monochrome)
DH = Row # of cursor
DL = Column # of cursor

OUTPUT: None

READ CURSOR POSITION

INPUT: BH = ACTIVE DISPLAY PAGE
(ignored and set to 0 if graphics or monochrome mode)

RETURNED: DH = ROW LOCATION OF CURSOR
DL = COLUMN LOCATION OF CURSOR
CX = CURSOR TYPE

OUTPUT: AX = Undefined (however, we return it unchanged)

READ LIGHT PEN (AH = 04H)

INPUT: None

OUTPUT: AH = 0 if light pen not triggered, 1 if it is
DH = Character row of light pen
DL = Character column of light pen
CH = Pixel row
BX = Pixel column, best estimate

SELECT ACTIVE DISPLAY PAGE (AH = 05H)

INPUT: AL = NEW ACTIVE DISPLAY PAGE

OUTPUT: AX = (?)

SCROLL ACTIVE PAGE UP (AH = 06H)

INPUT: AL = LINES TO SCROLL (CLEAR WINDOW IF 0)
BH = ATTRIBUTE FOR BLANK LINE(S)
CH, CL = ROW/COLUMN OF UPPER LEFT CORNER OF WINDOW
DH, DL = ROW/COLUMN OF LOWER RIGHT CORNER OF WINDOW

OUTPUT: None

SCROLL ACTIVE PAGE DOWN (AH = 07H)

INPUT: AL = LINES TO SCROLL (CLEAR WINDOW IF 0)
BH = ATTRIBUTE FOR BLANK LINE(S)
CH, CL = ROW/COLUMN OF UPPER LEFT CORNER OF WINDOW
DH, DL = ROW/COLUMN OF LOWER RIGHT CORNER OF WINDOW

OUTPUT: None

READ CHAR & ATTRIBUTE AT CURSOR POSITION (AH = 08H)

INPUT: BH = ACTIVE DISPLAY PAGE

OUTPUT: AL = CHARACTER
AH = ATTRIBUTE
(not defined for graphics, however we return an ORing of any and all color bits set as the attribute, a reasonable compromise)

GRAPHICS MODE READ:

OUTPUT: AL = CHAR READ (if recognized, else 0)
AH = ATTRIBUTE (COLOR) (If recognized, else 0)
All characters above 80h are recognized if the RAM font vector is other than 0, else not

WRITE CHAR & ATTRIBUTE AT CURSOR POSITION (AH = 09H)

INPUT: BH = ACTIVE DISPLAY PAGE
CX = NUMBER OF TIMES TO WRITE CHARACTER
AL = CHAR TO WRITE
BL = CHARACTER ATTRIBUTE

OUTPUT: None

WRITE CHAR AT CURSOR POSITION (AH = 0AH)

INPUT: BH = ACTIVE DISPLAY PAGE
CX = NUMBER OF TIMES TO WRITE CHARACTER
AL = CHAR TO WRITE
BL = Character Attribute if in a graphics mode otherwise ignored

OUTPUT: None

SET COLOR PALETTE (AH = 0BH)

INPUT: BH = 0 FOR BACKGROUND COLOR IN BL
1 FOR COLOR SET NUMBER IN BL
BL = BITS 0-4 IF BH = 0
0 FOR COLOR SET GREEN/RED/YELLOW
F BH = 1
1 FOR COLOR SET CYAN/MAGENTA/WHITE
IF BH = 1

WRITE DOT (AH = 0CH)

INPUT: DX = ROW NUMBER (MODE DEPENDENT)
CX = COLUMN NUMBER (MODE DEPENDENT)
AL = COLOR VALUE

OUTPUT: AH = ?

READ DOT (AH = 0DH)

INPUT: DX = ROW NUMBER (MODE DEPENDENT)
CX = COLUMN NUMBER (MODE DEPENDENT)

OUTPUT: AH = ?
AL = COLOR VALUE

WRITE TELETYPE (AH = OEH)

INPUT AL = CHARACTER TO BE WRITTEN
BL = FOREGROUND COLOR OF CHAR (USED ONLY IN GRAPHICS MODE)
BH = REQUESTED DISPLAY PAGE (REALLY IS IGNORED)

OUTPUT:None

READ CURRENT VIDEO STATE (AH = OFH)

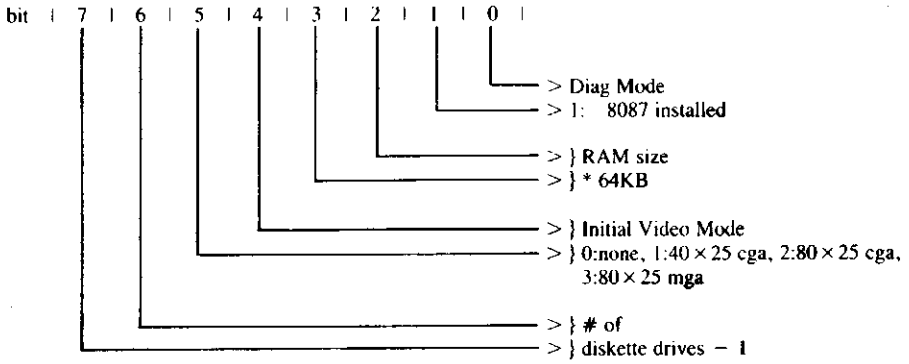
INPUT: DS = ROM data segment

OUTPUT: AH = NUMBER OF SCREEN COLUMNS
AL = CURRENT VIDEO MODE
BH = ACTIVE DISPLAY PAGE

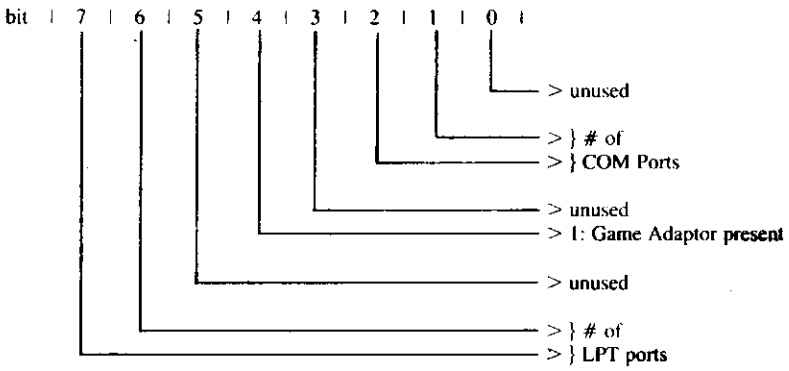
EQUIPMENT CHECK VIA S/W INT 11H

OUTPUT: AX = Equipment Flags

Bits of AL:



Bits of AH:



MEMORY SIZE CHECK VIA S/W INT 12H

OUTPUT: AX = Total Memory size in Kilobytes

DISKDSR ENTRY POINT VIA S/W INT 13H

RESET DISK SUBSYSTEM (AH = 00H)

OUTPUT: AH = DISK STATUS

READ DISK STATUS (AH = 01H)

OUTPUT: AH & AL = DISK STATUS

READ SECTOR(S) (AH = 02H)
WRITE SECTOR(S) (AH = 03H)
VERIFY SECTOR(S) (AH = 04H)

INPUT: DL = DRIVE NUMBER (0-3)
DH = HEAD NUMBER (0-1)
CH = TRACK NUMBER (0-39)
CL = SECTOR NUMBER (1-8)
AL = NUMBER OF SECTORS TO READ, WRITE OR VERIFY (1-8)
ES:BX = BUFFER ADDRESS

OUTPUT: AH = DISK STATUS
AL = 0

FORMAT TRACK (AH = 05H)

INPUT: DL = DRIVE NUMBER (0-3)
DH = HEAD NUMBER (0-1)
CH = TRACK NUMBER (0-39)
AL = # of sectors to format to see if we have a DMA boundary error
ES:BX = BUFFER ADDRESS 4-BYTE TRACK INFO FIELDS (C,H,R,N):
C = TRACK NUMBER
H = HEAD NUMBER
R = SECTOR NUMBER
N = BYTES/SECTOR (00 = 128, 01 = 256, 10 = 512, 11 = 1024)

OUTPUT: AH = DISK STATUS

DISK STATUS RETURNED IN AH (IF CF = 1)

01H – Illegal Command
02H – Address Mark not Found
03H – Write Protect Error
04H – Sector not found
06H – No Diskette
08H – DMA Overrun
09H – DMA Boundary Violation
10H – CRC Error
20H – FDC Error
40H – Seek Error
80H – Timeout

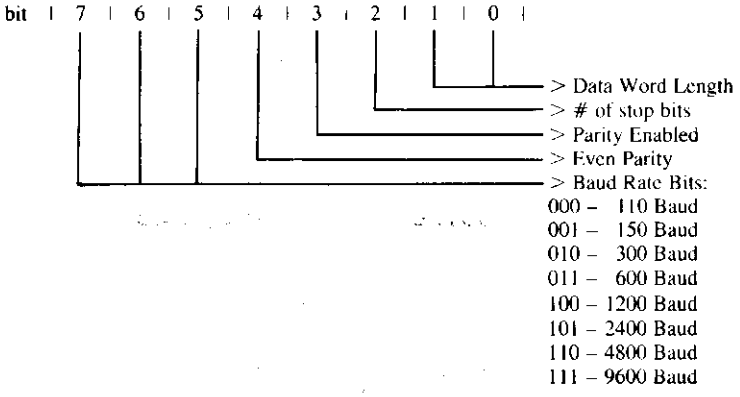
EIA DSR ENTRY POINT VIA S/W INT 14H

INITIALIZE COMM PORT (AH = 00H)

INPUT: DX = Modem Control Register port
AL = Baud Rate and UART control parameters
BH = 0, upper bits of baud rate index

OUTPUT: AH = Line Status
AL = Modem Status
Serial Port Control bits in AL Register

Bits of AL on Entry:



TRANSMIT A CHAR (AH = 01H)

INPUT: DX = Index into device table
AL = Character to transmit
CX = Timeout value
BX = 0, used as timeout counter

OUTPUT: AH = Line Status

RECEIVE A CHAR (AH = 02H)

INPUT: DX = Index into device table
CX = Timeout value
BX = 0, used as timeout counter

OUTPUT: AH = Line Status (error bits only, = 0 if OK)
AL = Received Character

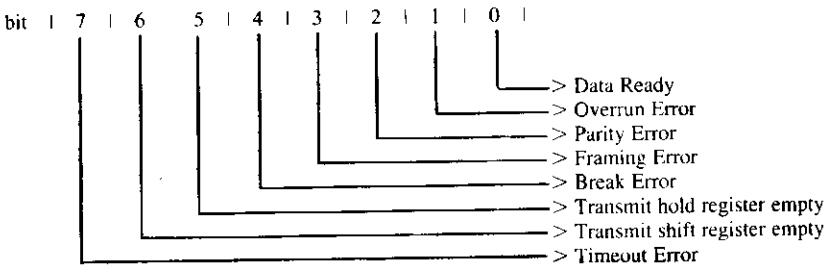
RETURN SERIAL PORT STATUS (AH = 03H)

INPUT: DX = Modem Control Register port

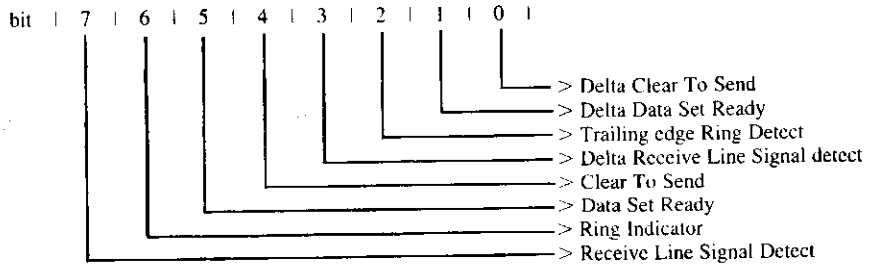
OUTPUT: AH = Line Status
AL = Modem Status

Serial Port Status bits returned in AX Register:

AH Register:



AH Register:



KYBDSR ENTRY POINT VIA S/W INT 16H

READ KEYBOARD INPUT (AH = 00H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = ASCII CHARACTER
AH = SCAN CODE

READ KEYBOARD STATUS (AH = 01H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = ASCII CHARACTER
AH = SCAN CODE
Z FLAG = 1 if no character available
Z FLAG = 0 if character available

READ SHIFT STATUS (AH = 02H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = SHIFT STATUS BYTE

CASSETTE INT 15H DSR

OUTPUT: AH = 86h, Error code, Carry set.
Interrupts off

KYBDSR ENTRY POINT VIA S/W INT 16H

READ KEYBOARD INPUT (AH = 00H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = ASCII CHARACTER
AH = SCAN CODE

READ KEYBOARD STATUS (AH = 01H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = ASCII CHARACTER
AH = SCAN CODE
Z FLAG = 1 if no character available
Z FLAG = 0 if character available

READ SHIFT STATUS (AH = 02H)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = SHIFT STATUS BYTE

PRINTER ENTRY POINT VIA S/W INT 17H

PRINT CHARACTER (AH = 00H)

INPUT: AL = Character to output
DX = Index to Printer table port + 1
(the Status port)
CX = Timeout value

OUTPUT: AH = Printer Status

INITIALIZE PRINTER (AH = 01H)

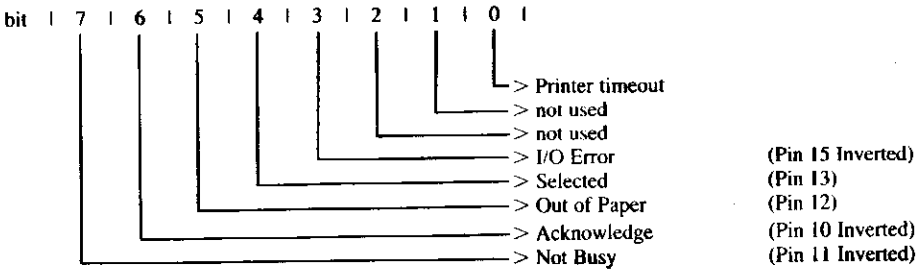
INPUT: DX = Printer Status Port address = (Printer Table contents + 1)

OUTPUT: AH = Printer Status

RETURN PRINTER STATUS (AH = 02H)

INPUT: DX = Index to Printer table port + 1 (the Status port)
CH must have correct value of timeout flag

OUTPUT: AH = Printer Status:



Notes: Pins #s are those on a 25 pin D connector

ROM BASIC ENTRY VIA S/W INT 18H

Not able to boot diskette, go to Monitor, error message or a ROM BASIC

BOOT FROM DISKETTE VIA S/W INT 19H

*** B O O T D I S K E T T E ***

If boot attempt fails:

Fall through to user routine INT 18h, which might be a monitor or a ROM BASIC, an error message etc.
Should INT 18h return, which is unlikely, we'll return to caller of INT 19h.

TIMER DEVICE SERVICE ROUTINE — INT 1Ah

READ CLOCK (AH = 00h)

INPUT: DS = ROM data segment (0040h)

OUTPUT: AL = 24-Hour Rollover flag
CX = High word of Clock Count
DX = Low word of Clock Count

SET CLOCK (AH = 01h)

INPUT: DS = ROM data segment (0040h)
CX = High word of Clock Count
DX = Low word of Clock Count

OUTPUT: AH = 0

USER SUPPLIED KEYBOARD BREAK ROUTINE — INT 1Bh

When a CTRL + ScrLock is detected, this INT is issued. A USER Break Routine may be invoked here. Note that this function is used by MSDOS.

USER SUPPLIED TIMER INTERVAL TICK — INT 1Ch

This interrupt is called internally after each timer interrupt (18.2 Hz). It is initialized to point to a dummy 1RET instruction.

CRT CONTROLLER PARAMETERS — DWORD POINTER AT INT IDH VECTOR

Default Video CRT Parameter Block

DEFAULT DISK PARAMETER BLOCK — DWORD POINTER AT INT 1EH VECTOR

Default Diskette Parameter Block

EXTENDED GRAPHIC CHARACTER SET — DWORD POINTER AT INT 1FH VECTOR

Character Generator ROM used in Graphic mode for characters 80H to OFFh.

Janus.Library

PREFACE

This is a brief description of the janus code. This code supports low level access to the "janus" system — the link between a PC and an Amiga.

THE PUBLIC ROUTINES

Contents

Descriptions

The code is packaged as a library (specifically "janus.library"), which is loaded during Autoconfig procedure.

All routines that return a value return it in D0. There is a link library for C routines, "jlib.lib".

oldHandler = SetJanusHandler(jintnum, intserver)
D0 A1

This routine sets up an interrupt handler for a particular janus interrupt. The old interrupt is returned. A null means that there is no interrupt handler. If there is no interrupt handler then interrupts not will be processed for that jintnum.

oldEnable = SetJanusEnable(jintnum, newvalue)
D0 D1

Each jintnum may be individually enabled or disabled (this is in addition to the control of setting the interrupt handler to NULL). If the interrupt is disabled then requests that are received will not generate interrupts. These requests may be detected via SetJanusRequest.

If newvalue is 0 then the interrupt is disabled. If it is 1 then the interrupt is enabled. All other values are reserved.

This routine will generate an interrupt if it an interrupt is enabled that has a pending request. This does not currently happen until the next hardware interrupt occurs.

oldRequest = SetJanusRequest(jintnum, newvalue)
D0 D1

This routine sets or clears an interrupt request for jintnum. If newvalue is zero then the request is cleared. If newvalue is one then the request is set. In either case the old value of the request is returned.

Setting a request will generate an interrupt (if it is enabled). This does not currently happen.

SendJanusInt(jintnum)
D0

This call is useful for "system" requests — e.g. those requests not directly defined by the hardware. The call marks the request in the system interrupt area and then posts a hardware interrupt to the pc.

CheckJanusInt(jintnum)
D0

This call returns the status byte from the interrupt area. It can be used to tell if the pc has noticed the interrupt yet. A value of JNOINT ($ff) means no interrupt is pending (which probably means that the pc has already processed it). JSENDINT ($7f) means that the interrupt is pending. Anything else should be treated with suspicion.

ptr = AllocJanusMem(size, type)
D0 D1

This routine allocates memory from the parameter or buffer memory free pools, and returns a 68000 addressable pointer to the memory. It allocates "size" bytes, or returns NULL if there was not enough memory.

The type field is used to determine which pool of memory is desired. It should be either MEMF_PARAMETER or MEMF_BUFFER. In addition, you may specify what sort of memory access the pointer should refer to. The four choices are MEM_BYTEACCESS, MEM_WORD-

ACCESS, MEM_GRAPHICACCESS, or MEM_IOACCESS. See the hardware description for the meaning of these access methods if you do not already know.

FreeJanusMem(ptr, size)
A1 D0

The specified memory is returned to the free pool. Some modest error checking is done, and the system will Alert if there is a problem.

ptr = JanusMemBase(type)
D0

The base of the memory referred to by the type specifier is returned. See AllocJanusMem for a (very) brief description of type.

type = JanusMemType(ptr)
D0

The type of the pointer is returned. "Unpredictable results" will occur if ptr points to neither buffer memory nor parameter memory.

offset = JanusMemToOffset(ptr)
D0

If ptr points to buffer or parameter memory, the offset from the start of that memory to ptr will be returned. This is the value that should be fed to SetParamOffset( ) if this is a parameter block.

offset = GetParamOffset(jintnum)
D0

The parameter offset for interrupt jintnum is returned. The system does not interpret this number, but by convention a $ffff means that no parameter block has been set up.

oldOffset = SetParamOffset(jintnum, offset)
D0 D1

The parameter offset for jintnum is set to the bottom sixteen bits of offset. The previous offset is returned. The system does not interpret this number, but by convention a $ffff means that no parameter block had previously been set up.

ptr = GetJanusStart( )

The base of the janus board is returned.

setupSig = SetupJanusSig
(jintnum, signum, paramsize, paramtype)
D0 D1 D2 D3

This routine does the "standard" things that most users of the janus system would want. It is conceivable that most people who use the janus board will use only this routine and CleanupJanusSig( ).

The main purpose is to set up an interrupt handler for your interrupt, and translate this into an exec signal that will be sent to your task. This allows you to ignore all the complexities of writing interrupt code.

You specify the jintnum to use as the interrupt number and the signal number (signum) to be sent to you. Jintnum should (for now) be gotten via the include file services.[hi]. Signum will most often be gotten via AllocSignal(-1), which allocates an unused signal.

In addition to setting up a way to get interrupts, the call can set up a parameter area. It will allocate paramsize bytes of type paramtype, and set up the parameter area to point to them.

There is some error checking done while all this is going on. If signum is -1 the call fails (-1 is the error return from AllocSignal...). If there is already an interrupt handler then the call fails. If paramsize is non-zero and there is already a parameter area the call fails. If it cannot allocate enough memory the call fails.

The call returns a NULL if it fails. If it succeeds then a pointer to a SetupSig structure is returned. This structure is defined in setupsig.[hi].

CleanupJanusSig(setsupSig)
A0

This routine undoes everything that SetupJanusSig does.

JanusLock(ptr)
A0

Gain a janus lock (e.g. a lock on a memory list). You must not keep this lock for a long time — keep it just long enough to manipulate the data structure associated with the lock, and don't go to sleep.

JanusUnLock(ptr)
A0

Release a janus lock.

JBCopy(source, designation, length)
A0 A1 D0

Copy arbitratily aligned memory as efficiently as possible with the processor.

INCLUDE FILES

janus.[hi]:
gives interface to janus.library. All definitions in this file are amiga specific. The most useful thing in this file are the definitions for janus memory allocation types

janusreg.[hi]:
hardware constants. Most people should not need this. If you do, we need to hide more information.

janusvar.[hi]:
the shared data structure between the amiga and the pc. Once again, you should not need direct access to these routines. We have tried to provide interface routines to do all the normal things.

i86block.i:
command blocks for calling pc's interrupt's directly and for the hard disk.

services.[hi]:
hard coded constants for interrupt numbers. Eventually these numbers will be gotten at run time, but for now they are constants. These numbers correspond to the "jintnum" parameters below.

setupsig.[hi]:
data structure for SetupJanusSig( ) call.

LISTINGS

```
i86block.i — interface definitions between amiga and
             commodore-pc
Copyright © 1986, Commodore-Amiga Inc., All rights reserved

        IFND    JANUS_I86BLOCK_I
JANUS_I86BLOCK_I        SET 1

; All registers in this section are arranged to be read and written
; from the WordAccessOffset area of the shared memory. If you really
; need to use the ByteAccessArea, all the words will need to be byte
; swapped.
```

```
; Syscall86 — how the 8086/8088 wants its parameter block arranged:

STRUCTURE Syscall86,0
    UWORD   s86_AX
    UWORD   s86_BX
    UWORD   s86_CX
    UWORD   s86_DX
    UWORD   s86_SI
    UWORD   s86_DS
    UWORD   s86_DI
    UWORD   s86_ES
    UWORD   s86_BP
    UWORD   s86_PSW
    UWORD   s86_INT             ; 8086 int # that will be
                                  called
    LABEL   Syscall86_SIZEOF

; Syscall68 — the way the 68000 wants its parameters arranged:

STRUCTURE Syscall68,0
    ULONG   s68_D0
    ULONG   s68_D1
    ULONG   s68_D2
    ULONG   s68_D3
    ULONG   s68_D4
    ULONG   s68_D5
    ULONG   s68_D6
    ULONG   s68_D7
    ULONG   s68_A0
    ULONG   s68_A1
    ULONG   s68_A2
    ULONG   s68_A3
    ULONG   s68_A4
    ULONG   s68_A5
    ULONG   s68_A6
    ULONG   s68_PC              ; pc to start execution from
    ULONG   s68_ArgStack        ; array to be pushed onto
                                  stack
    ULONG   s68_ArgLength       ; number of bytes to be
                                  pushed (must be even)
    ULONG   s68_MinStack        ; minimum necessary stack
                                  (0 = use default)
    ULONG   s68_CCR             ; condition code register
    ULONG   s68_Process         ; ptr to process for this
                                  block.
    UWORD   s68_Command         ; special commands: see
                                  below
    UWORD   s68_Status          ;
    UWORD   s68_SigNum          ; internal use: signal to
                                  wake up process
```

```
LABEL       Syscall68_SIZEOF

S68COM_DOCALL       EQU 0       ; normal case _jsr to specified Program cntr
S68COM_REMPROC      EQU 1       ; kill process
S68COM_CRPROC       EQU 2       ; create the process, but do not call anything

; Disk request structure for raw amiga access to 8086's disk
; goes directly to PC BIOS (via PC int 13 scheduler):

STRUCTURE DskAbsReq,0
  UWORD   dar_FktCode           ; bios function code (see ibm tech ref)
  UWORD   dar_Count             ; sector count
  UWORD   dar_Track             ; cylinder #
  UWORD   dar_Sector            ; sector #
  UWORD   dar_Drive             ; drive
  UWORD   dar_Head              ; head
  UWORD   dar_Offset            ; offset of buffer in MEMF_BUFFER memory
  UWORD   dar_Status            ; return status
  LABEL   DskAbsReq_SIZEOF

; Definition of an AMIGA disk partition. returned by info function:

STRUCTURE DskPartition,0
  UWORD   dp_Next               ; 8088 ptr to next part. 0 -> end of list
  UWORD   dp_BaseCyl            ; cyl # where partition starts
  UWORD   dp_EndCyl             ; last cyclinder # of this partition
  UWORD   dp_DrvNum             ; DOS drive number (80H, 81H, ...)
  UWORD   dp_NumHeads           ; number of heads for this drive
  UWORD   dp_NumSecs            ; number of sectors per track for this drive
  LABEL   DskPartition_SIZEOF

; Disk request structure for higher level Amiga disk request to 8086:

STRUCTURE AmigaDskReq,0
  UWORD   adr_Fnctn             ; function code (see below)
  UWORD   adr_Part              ; partition number (0 is first partition)
  ULONG   adr_Offset            ; byte offset into partition
  ULONG   adr_Count             ; number of bytes to transfer
```

```
    UWORD    adr_BufferAddr          ; offset into MEMF-
                                       _BUFFER memory for
                                       buffer
    UWORD    adr_Err                 ; return code, 0 if all OK
LABEL        AmigaDskReq_SIZEOF

; Function codes for AmigaDskReq adr_Fnctn word:

ADR_FNCTN_INIT      EQU   0   ; given nothings, sets adr_
                              Part to # partitions
ADR_FNCTN_READ      EQU   1   ; given partition, offset,
                              count, buffer
ADR_FNCTN_WRITE     EQU   2   ; given partition, offset,
                              count, buffer
ADR_FNCTN_SEEK      EQU   3   ; given partition, offset
ADR_FNCTN_INFO      EQU   4   ; given part, buff adr, cnt,
                              copys in a DskPartition
                              structure. cnt set to actual
                              number of bytes copied.

; Error codes for adr_Err, returned in low byte:

ADR_ERR_OK          EQU   0   ; no error
ADR_ERR_OFFSET      EQU   1   ; offset not on sector
                              boundary
ADR_ERR_COUNT       EQU   2   ; dsk_count not a multiple
                              of sector size
ADR_ERR_PART        EQU   3   ; partition does not exist
ADR_ERR_FNCT        EQU   4   ; illegal function code
ADR_ERR_EOF         EQU   5   ; offset past end of
                              partition
ADR_ERR_MULPL       EQU   6   ; multiple calls while
                              pending service

; Error condition from IBM-PC BIOS, returned in high byte:

ADR_ERR_SENSE_FAIL          EQU  $ff
ADR_ERR_UNDEF_ERR           EQU  $bb
ADR_ERR_TIME_OUT            EQU  $80
ADR_ERR_BAD_SEEK            EQU  $40
ADR_ERR_BAD_CNTRLR          EQU  $20
ADR_ERR_DATA_CORRECTED      EQU  $11  ; data corrected
ADR_ERR_BAD_ECC             EQU  $10
ADR_ERR_BAD_TRACK           EQU  $0b
ADR_ERR_DMA_BOUNDARY        EQU  $09
ADR_ERR_INIT_FAIL           EQU  $07
ADR_ERR_BAD_RESET           EQU  $05
ADR_ERR_RECRD_NOT_FOUND     EQU  $04
ADR_ERR_BAD_ADDR_MARK       EQU  $02
ADR_ERR_BAD_CMD             EQU  $01

ENDC     !JANUS_I86BLOCK_I
```

```
janus.i — software conventions for janus.i
Copyright © 1986, Commodore-Amiga Inc., All rights reserved

IFND    EXEC_TYPES_I
INCLUDE "exec/types.i"
ENDC    EXEC_TYPES_I

IFND    EXEC_LIBRARIES_I
        INCLUDE "exec/libraries.i"
ENDC    EXEC_LIBRARIES_I

IFND    EXEC_INTERRUPTS_I
INCLUDE "exec/interrupts.i"
ENDC    EXEC_INTERRUPTS_I

; JanusResource — an entity which keeps track of the reset state of
  the 8088. If this resource does not exist, it is assumed the 8088 can
  be reset.

STRUCTURE JanusResource,LN_SIZE
    APTR    jr_BoardAddress         ; address of JANUS board
    UBYTE   jr_Reset                ; non_zero indicates 8088
                                      is held reset
    LABEL   JanusResource_SIZEOF

; As a coding convenience, we assume a maximum of 32 handlers.
; People should avoid using this in their code, because we want to
; be able to relax this constraint in the future. All the standard
; commands' syntactically support any number of interrupts, but
; the internals are limited to 32.

MAXHANDLER      EQU     32

; JanusAmiga — amiga specific data structures for janus project:

STRUCTURE JanusAmiga,LIB_SIZE
ULONG   ja_IntReq               ; software copy of out-
                                  standing requests
ULONG   ja_IntEna               ; software copy of enabled
                                  interrupts
APTR    ja_ParamMem             ; ptr to (word arranged)
                                  param mem
APTR    ja_IoBase               ; ptr to base of io register
                                  region
APTR    ja_ExpanBase            ; ptr to start of shared
                                  memory
APTR    ja_ExecBase             ; ptr to exec library
APTR    ja_DOSBase              ; ptr to DOS library
APTR    ja_SegList              ; holds a pointer to our
                                  code segment
```

```
APTR        ja_IntHandlers              ; base of array of int server
                                          ptrs
STRUCT      ja_IntServer,IS_SIZE        ; INTB_PORTS server
STRUCT      ja_ReadHandler,IS_SIZE      ; JSERV_READAMIGA
                                          handler
LABEL       JanusAmiga_SIZEOF

; Hide a byte quantity in the lib_pad field
ja_SpuriousMask   EQU   LIB_pad

; Magic constants for memory allocation:

MEM_TYPEMASK        EQU  $00ff  ; 8 memory areas
BITDEF        MEM,PARAMETER,0   ; parameter memory
BITDEF        MEM,BUFFER,1      ; buffer memory
MEM_ACCESSMASK      EQU  $3000  ; bits that participate in
                                  access types
MEM_BYTEACCESS      EQU  $0000  ; return base suitable for byte
                                  access
MEM_WORDACCESS      EQU  $1000  ; return base suitable for
                                  word access
MEM_GRAPHICACCESS EQU  $2000  ; return base suitable for
                                  graphic access
MEM_IOACCESS        EQU  $3000  ; return base suitable for
                                  io access
TYPEACCESSTOADDR    EQU  5      ; # of bits to change access
                                  mask into addr

; Macro to lock access to janus data structures from PC side:
LOCK        MACRO       ; ( 1 — effective address of lock byte )
begin@
            tas         1
            beq.s       exit@
            nop
            nop
            bra.s       begin@
exit@:
            endm

UNLOCK      MACRO       ; ( 1 — effective address of lock byte )
            move.b      #0,1
            ENDM

JANUSNAME   MACRO
            dc.b        'janus.library',0
            ENDM
```

```
; Hardware interrupt bits (all bits are active low)
BITDEF   JINT,MINT,0      ; mono video ram written to
BITDEF   JINT,GINT,1      ; color video ram written to
BITDEF   JINT,CRT1INT,2   ; mono video control registers changed
BITDEF   JINT,CRT2INT,3   ; color video control registers changed
BITDEF   JINT,ENBKB,4     ; keyboard ready for next character
BITDEF   JINT,LPT1INT,5   ; parallel control register
BITDEF   JINT,COM2INT,6   ; serial control register
BITDEF   JINT,SYSINT,7    ; software int request

; The Amiga side of the Bridgeboard has four sections of its address
; space. Three of these parts are different arrangements of the same
; memory. The fourth part has the specific amiga accessible I/O
; registers (jio—??). The other three parts all contain the same
; data, but the data is arranged in different ways: Byte Access
; lets the 68k read byte streams written by the 8088, Word Access
; lets the 68k read word streams written by the 8088, and Graphic
; Access lets the 68k read medium res graphics memory in a more
; efficient manner (the pc uses packed two bit pixels; graphic
; access rearranges these data bits into two bytes, one for each bit
; plane).

ByteAccessOffset        EQU $00000
WordAccessOffset        EQU $20000
GraphicAccessOffset     EQU $40000
IoAccessOffset          EQU $60000

; Within each bank of memory are several sub regions. These are the
; definitions for the sub regions:

BufferOffset            EQU $00000
ColorOffset             EQU $10000
ParameterOffset         EQU $18000
MonoVideoOffset         EQU $1c000
IoRegOffset             EQU $1e000

BufferSize              EQU $10000
ParameterSize           EQU $04000
```

These are the definitions for the io registers. All the registers are byte wide and the address are for Byte Access addresses:

```
jio-KeyboardData        EQU  $061f ; data that keyboard will read
jio_SystemStatus        EQU  $003f ; pc only register

jio_NmiEnable           EQU $005f  ; pc only register

jio_Com2XmitData        EQU $007d
jio_Com2ReceiveData     EQU $009d
jio_Com2IntEnableW      EQU $00bd
```

```
jio_Com2IntEnableR    EQU $00dd
jio_Com2DivisorLSB    EQU $007f
jio_Com2DivisorMSB    EQU $009f
jio_Com2IntID         EQU $00ff
jio_Com2LineCntrl     EQU $011f
jio_Com2ModemCntrl    EQU $013f
jio_Com2LineStatus    EQU $015f
jio_Com2ModemStatus   EQU $017f

jio_Lpt1Data          EQU $019f  ; data byte
jio_Lpt1Status        EQU $01bf  ; see equates below
jio_Lpt1Control       EQU $01df  ; see equates below

jio_MonoAddressInd    EQU $01ff  ; current index into crt data regs
jio_MonoData          EQU $02a1  ; every other byte for 16 registers
jio_MonoControlReg    EQU $02ff

jio_ColorAddressInd   EQU $021f  ; current index into crt data regs
jio_ColorData         EQU $02c1  ; every other byte for 16 registers
jio_ColorControlReg   EQU $023f
jio_ColorSelectReg    EQU $025f
jio_ColorStatusReg    EQU $027f

jio_DisplaySystemReg  EQU $029f

jio_IntReq            EQU $1ff1  ; read clears, pc -> amiga ints
jio_PcIntReq          EQU $1ff3  ; r/o, amiga -> pc ints
jio_ReleasePcReset    EQU $1ff5  ; r/o, strobe release pc's reset
jio_RamSize           EQU $1ff7  ; r/o, give ram addresses
jio_IntEna            EQU $1ff9  ; r/w, enables pc int lines
jio_PcIntGen          EQU $1ffb  ; w/o, bit = = 0 -> cause pc int
jio_Control           EQU $1ffd  ; w/o, random control lines
jio_RamBaseAddr       EQU $1fff  ; r/w, sets expansion ram base
                                   address

; Now the magic bits in each register (and boy, are there a lot of
  them!)

; Bits for Lpt1Status register
BITDEF    JPCLS,STROBE,0
BITDEF    JPCLS,AUTOFEED,1
BITDEF    JPCLS,INIT,2
BITDEF    JPCLS,SELECTIN,3
BITDEF    JPCLS,IRQENABLE,4   ; active 1

; Bits for Lpt1Control register
BITDEF
JPCLC,ERROR,3
BITDEF
JPCLC,SELECT,4
```

```
BITDEF      JPCLC,NOPAPER,5
BITDEF    JPCLC,ACK,6
BITDEF    JPCLC,BUSY,7

; Bits for PcIntReq, PcIntGen registers
BITDEF    JPCINT,IRQ1,0     ; active high
BITDEF    JPCINT,IRQ3,1     ; active low
BITDEF    JPCINT,IRQ4,2     ; active low
BITDEF    JPCINT,IRQ7,3     ; active low

; PC side interrupts
JPCKEYINT     EQU $ff     ; keycode available
JPCSENDINT    EQU $fc     ; system request
JPCLPT1INT    EQU $f6     ; printer acknowledge

; Bits for RamSize
BITDEF      JRAM,EXISTS,0     ; unset if there is any ram at all
BITDEF      JRAM,2MEG,1       ; set if 2 meg, clear if 1/2 meg

; Bits for control register
BITDEF  JCNTRL,ENABLEINT,0    ; enable amiga interrupts
BITDEF  JCNTRL,DISABLEINT,1   ; disable amiga interrupts
BITDEF  JCNTRL,RESETPC,2      ; reset the pc. remember to strobe
                              ;   ReleasePcReset afterwards
BITDEF  JCNTRL,CLRPCINT,3     ; turn off all amiga->pc ints (except
                              ;   keyboard

; Constants for sizes of various janus regions
JANUSTOTALSIZE      EQU 512*1024    ; 1/2 megabyte
JANUSBANKSIZE       EQU 128*1024    ; 128K per memory bank
JANUSNUMBANKS       EQU 4           ; four memory banks
JANUSBANKMASK       EQU $60000      ; mask bits for bank region
```

janusvar.i—the software data structure for the janus board


```
; All bytes described here are described in the byte order of the
; 8088. Note that words and longwords in these structures will be
; accessed from the word access space to preserve the byte order in
; a word — the 8088 will access longwords by reversing the words :
; like a 68000 access to the word access memory

; JanusMemHead — a data structure roughly analogous to an exec
; mem chunk. It is used to keep track of memory used between the
; 8088 and the 68000.
```

```
STRUCTURE JanusMemHead,0
        UBYTE   jmh_Lock            ; lock byte between processors
        UBYTE   jmh_pad0
        APTR    jmh_68000Base       ; rptr's are relative to this
        UWORD   jmh_8088Segment     ; segment base for 8088
        RPTR    jmh_First           ; offset to first free chunk
        RPTR    jmh_Max             ; max allowable index
        UWORD   jmh_Free            ; total number of free bytes -1
        LABEL   JanusMemHead_SIZEOF

STRUCTURE JanusMemChunk,0
        RPTR    jmc_Next    ; rptr to next free chunk
        UWORD   jmc_Size    ; size of chunk -1
        LABEL   JanusMemChunk_SIZEOF

STRUCTURE JanusBase,0
        UBYTE   jb_Lock         ; also used to handshake at 8088 reset
        UBYTE   jb_8088Go       ; unlocked to signal 8088 to initialize
        STRUCT  jb_ParamMem,JanusMemHead_SIZEOF
        STRUCT  jb_BufferMem,JanusMemHead_SIZEOF
        RPTR    jb_Interrupts
        RPTR    jb_Parameters
        UWORD   jb_NumInterrupts
        LABEL   JanusBase_SIZEOF

;------ constant to set to indicate a pending software interrupt
JSETINT         EQU     $7f

        FUNCDEF SetJanusHandler
        FUNCDEF SetJanusEnable
        FUNCDEF SetJanusRequest
        FUNCDEF SendJanusInt
        FUNCDEF CheckJanusInt
        FUNCDEF AllocJanusMem
        FUNCDEF FreeJanusMem
        FUNCDEF JanusMemBase
        FUNCDEF JanusMemType
        FUNCDEF JanusMemToOffset
        FUNCDEF GetParamOffset
        FUNCDEF SetParamOffset
        FUNCDEF GetJanusStart
        FUNCDEF SetupJanusSig
        FUNCDEF CleanupJanusSig
        FUNCDEF JanusLock
        FUNCDEF JanusUnLock
        FUNCDEF JBCopy
```

```
memrw.i—parameter area definition for access to other
        processors mem
```


```
    IFND        JANUS_MEMRW_I
JANUS_MEMRW_I   SET 1

; this is the parameter block for the JSERV_READPC and JSERV_
; READAMIGA; services — read and/or write the other processors
; memory.

STRUCTURE MemReadWrite,0
UWORD  mrw_Command      ; see below for list of commands
UWORD  mrw_Count        ; number of bytes to transfer
ULONG  mrw_Address      ; local address to access. This is
                        ;   a machine pointer for the 68000, and
                        ;   a segment/offset pair for the 808x.
                        ;   The address is arranged so the native
                        ;   processor may read it directly.
UWORD  mrw_Buffer       ; The offset in buffer memory for the
                        ;   other buffer.
UWORD  mrw_Status       ; See below for status.
LABEL  MemReadWrite_SIZEOF

; Command definitions:
MRWC_NOP        EQU 0 ; do nothing — return OK status
                      code
MRWC_READ       EQU 1 ; xfer from address to buffer
MRWC_WRITE      EQU 2 ; xfer from buffer to address
MRWC_READIO     EQU 3 ; only on 808x — read from IO
                      space
MRWC_WRITEIO    EQU 4 ; only on 808x — write to IO space
MRWC_WRITEREAD  EQU 5 ; write from buffer, then read back

; Status definitions:
MRWS_INPROGRESS EQU $ffff  ; we've noticed command and
                           are working on it
MRWS_OK         EQU $0000  ; completed OK
MRWS_ACCESSERR  EQU $0001  ; some sort of protection
                           violation
MRWS_BADCMD     EQU $0002  ; command that the server
                           doesn't understand

        ENDC

    IFND        JANUS_SERVICES_I
JANUS_SERVICES_I    EQU     1
```

```
memrw.i—parameter area definition for access to other
        processors mem
Copyright © 1986, Commodore-Amiga Inc., All rights reserved
```

```
; this is the table of hard coded services. Other services may exist
; that are dynamically allocated via AllocJanusService.

; Service numbers constrained by hardware:
JSERV_MINT          EQU 0   ; monochrome display written to
JSERV_GINT          EQU 1   ; color display written to
JSERV_CRT1INT       EQU 2   ; mono display's control registers
                              changed
JSERV_CRT2INT       EQU 3   ; color display's control registers
                              changed
JSERV_ENBKB         EQU 4   ; keyboard ready for next character
JSERV_LPT1INT       EQU 5   ; parallel control register
JSERV_COM2INT       EQU 6   ; serial control register
; hard coded service numbers
JSERV_PCBOOTED      EQU 7   ; PC is ready to service soft
                              interrupts

JSERV_SCROLL        EQU 8   ; PC is scrolling its screen
JSERV_HARDDISK      EQU 9   ; Amiga reading PC hard disk
JSERV_READAMIGA     EQU 10  ; PC reading Amiga mem
JSERV_READPC        EQU 11  ; Amiga reading PC mem
JSERV_AMIGACALL     EQU 12  ; PC executing Amiga subroutine
JSERV_PCCALL        EQU 13  ; Amiga causing PC interrupt
JSERV_NEWASERV      EQU 14  ; PC initiating Amiga side of a new
                              service
JSERV_NEWPCSERV EQU 15      ; Amiga initiating PC side of a new
                              service

                ENDC    JANUS_SERVICES_I

        IFND    JANUS_SETUPSIG_I
JANUS_SETUPSIG_I    EQU 1
```

```
setupsig.i—data structure for SetupJanusSig() routine
Copyright © 1986, Commodore-Amiga Inc., All rights reserved
```

```
                IFND        EXEC_TYPES_I
                INCLUDE     'exec/types.i'
                ENDC

                IFND        EXEC_INTERRUPTS_I
                INCLUDE     'exec/interrupts.i'
                ENDC
```

```
STRUCTURE SetupSig,IS_SIZE
        APTR        ss_TaskPtr
        ULONG       ss_SigMask
        APTR        ss_ParamPtr
        ULONG       ss_ParamSize
        UWORD       ss_JanusIntNum
        LABEL       SetupSig_SIZEOF

        ENDC
```



```
#ifndef     EXEC_TYPES_I
#include    "exec/types.h"
#endif      EXEC_TYPES_I

#ifndef     EXEC_LIBRARIES_I
#include    "exec/libraries.h"
#endif      EXEC_LIBRARIES_I

#ifndef     EXEC_INTERRUPTS_I
#include    "exec/interrupts.h"
#endif      EXEC_INTERRUPTS_I

/*
** As a coding convenience, we assume a maximum of 32 handlers.
** People should avoid using this in their code, because we want
** to be able to relax this constraint in the future. All the
** standard commands' syntactically support any number of
   interrupts,
** but the internals are limited to 32.
*/

#define MAXHANDLER 32

typedef     UWORD       RPTR;

/* JanusAmiga — amiga specific data structures for janus project */

struct JanusAmiga [
        struct Library ja_LibNode;
        ULONG   ja_IntReq;      /* software copy of outstanding
                                   requests */
        ULONG   ja_IntEna;      /* software copy of enabled
                                   interrupts */
        UBYTE   *ja_ParamMem;   /* ptr to (byte arranged) param
                                   mem */
```

```
    UBYTE   *ja_IoBase;       /* ptr to base of io register region */
    UBYTE   *ja_ExpanBase;    /* ptr to start of shared memory */
    APTR    ja_ExecBase;      /* ptr to exec library */
    APTR    ja_SegList;       /* ptr to loaded code */
    struct Interrupt **ja_IntHandlers;/* base of array of int
    handler ptrs */
    struct Interrupt ja_IntServer; /* INTB_PORTS server */
    struct Interrupt ja_ReadHandler;/* JSERV_READAMIGA
    handler */
];
/* hide a byte field in the lib_pad field */
#define   ja_SpurriousMask        lib_pad

/* magic constants for memory allocation */
#define MEM_TYPEMASK        0x00ff  /* 8 memory areas */
#define MEMB_PARAMETER      (0)     /* parameter memory */
#define MEMB_BUFFER         (1)     /* buffer memory */

#define MEMF_PARAMETER      (1<<0) /* parameter memory */
#define MEMF_BUFFER         (1<<1) /* buffer memory */

#define MEM_ACCESSMASK      0x3000  /* bits that participate in
                                       access types */
#define MEM_BYTEACCESS      0x0000  /* return base suitable for
                                       byte access */
#define MEM_WORDACCESS      0x1000  /* return base suitable for
                                       word access */
#define MEM_GRAPHICACCESS 0x2000  /* return base suitable for
                                       graphic access */
#define MEM_IOACCESS        0x3000  /* return base suitable for
                                       io access */

#define TYPEACCESSTOADDR  5         /* # of bits to turn access
                                       mask to addr */

#define JANUSNAME       "janus.library"
```

janusreg.h — janus hardware registers (from amiga point of view)


```
/* hardware interrupt bits all bits are active low */
#define JINTB_MINT      (0)     /* mono video ram written to */
#define JINTB_GINT      (1)     /* color video ram written to */
#define JINTB_CRT1INT   (2)     /* mono video control registers
                                   changed */
#define JINTB_CRT2INT   (3)     /* color video control registers
                                   changed */
#define JINTB_ENBKB     (4)     /* keyboard ready for next
                                   character */
```

```
#define JINTB_LPT1INT   (5)     /* parallel control register */
#define JINTB_COM2INT   (6)     /* serial control register */
#define JINTB_SYSINT    (7)     /* software int request */

#define JINTF_MINT      (1<<0)
#define JINTF_GINT      (1<<1)
#define JINTF_CRT1INT   (1<<2)
#define JINTF_CRT2INT   (1<<3)
#define JINTF_ENBKB     (1<<4)
#define JINTF_LPT1INT   (1<<5)
#define JINTF_COM2INT   (1<<6)
#define JINTF_SYSINT    (1<<7)

/*
** The amiga side of the janus board has four sections of its address space.
** Three of these parts are different arrangements of the same memory. The
** fourth part has the specific amiga accessible IO registers (jio--??).
** The other three parts all contain the same data, but the data is arranged
** in different ways: Byte Access lets the 68k read byte streams written
** by the 8088, Word Access lets the 68k read word streams written by the
** 8088, and Graphic Access lets the 68k read medium res graphics memory
** in a more efficient manner (the pc uses packed two bit pixels; graphic
* access rearranges these data bits into two bytes, one for each bit plane).
*/

#define ByteAccessOffset        0x00000
#define WordAccessOffset        0x20000
#define GraphicAccessOffset     0x40000
#define IoAccessOffset          0x60000

#define jio_IntReq          0x1ff1  /* read clears, pc -> amiga ints */
#define jio_PcIntReq        0x1ff3  /* r/o, amiga -> pc ints */
#define jio_ReleasePcReset  0x1ff5  /* r/o, strobe release pc's reset */
#define jio_RamSize         0x1ff7  /* r/o, give ram addresses */
#define jio_IntEna          0x1ff9  /* r/w, enables pc int lines */
#define jio_PcIntGen        0x1ffb  /* w/o, bit == 0 -> cause pc int */
#define jio_Control         0x1ffd  /* w/o, random control lines */
```

```
#define jio_RamBaseAddr   0x1fff      /* r/w, sets extra ram base
                                         address */

/* now the magic bits in each register (and boy, are there a lot of
them!) */

/* bits for PcIntReq, PcIntGen registers */
#define JPCINTB_IRQ1      (0)         /* active high */
#define JPCINTB_IRQ3      (1)         /* active low */
#define JPCINTB_IRQ4      (2)         /* active low */
#define JPCINTB_IRQ7      (3)         /* active low */

#define JPCINTF_IRQ1      (1<<0)
#define JPCINTF_IRQ3      (1<<1)
#define JPCINTF_IRQ4      (1<<2)
#define JPCINTF_IRQ7      (1<<3)

/* pc side interrupts */
#define JPCKEYINT         (0xff)      /* keycode available */
#define JPCSENDINT        (0xfc)      /* system request */
#define JPCLPT1INT        (0xf6)      /* parallel port acknowledge */

/* bits for RamSize */
#define JRAMB_EXISTS      (0)         /* set if there is any ram at
                                         all */
#define JRAMB_2MEG        (1)         /* set if 2 meg, clear if 1/2
                                         meg */

#define JRAMF_EXISTS      (1<<0)
#define JRAMF_2MEG        (1<<1)

/* bits for control register */
#define JCNTRLB_ENABLEINT   (0)       /* enable amiga interrupts */
#define JCNTRLB_DISABLEINT  (1)       /* disable amiga interrupts */
#define JCNTRLB_RESETPC     (2)       /* reset the pc. remember to
                                         strobe */
                                      /* ReleasePcReset afterwards */
#define JCNTRLB_CLRPCINT    (3)       /* turn off all amiga->pc ints */

/* constants for sizes of various janus regions */
#define JANUSTOTALSIZE    (512*1024)  /* 1/2 megabyte */
#define JANUSBANKSIZE     (128*1024)  /* 128K per memory
                                         bank */
#define JANUSNUMBANKS     (4)         /* four memory
                                         banks */
#define JANUSBANKMASK     (0x60000)   /* mask bits for bank
                                         region */
```

janus.h—the software data structures for the janus board


```
/* all bytes described here are described in the byte order of the
 * 8088. Note that words and longwords in these structures will be
 * accessed from the word access space to preserve the byte order in
 * a word — the 8088 will access longwords by reversing the words:
 * like a 68000 access to the word access memory.
 */
/* JanusMemHead — a data structure roughly analogous to an exec
  mem chunk.
 * It is used to keep track of memory used between the 8088 and the
  68000.
 */

struct JanusMemHead [
        UBYTE   jmh_Lock;           /* lock byte between
                                       processors */
        UBYTE   jmh_pad0;
        APTR    jmh_68000Base;      /* rptr's are relative to this */
        UWORD   jmh_8088Segment;/* segment base for 8088 */
        RPTR    jmh_First;          /* offset to first free chunk */
        RPTR    jmh_Max;            /* max allowable index */
        UWORD   jmh_Free;           /* total number of free
                                       bytes -1 */
];

/* JanusMemChunk — keep track of individualy freed chunks of
memory.
 * Memory Chunks are longword aligned in this memory.
 */

struct JanusMemChunk [
        RPTR    jmc_Next;           /* rptr to next free chunk */
        UWORD   jmc_Size;           /* size of chunk -1*/
];

#ifdef undef
this stuff is saved for future use, but is not yet thought out
/* JanusList — an RPTR/Exec style list header.
 */
struct JanusList [
        RPTR    jl—Head;
        RPTR    jl—Tail;
        RPTR    jl—TailPred;
        UBYTE   jl—Lock;            /* lock byte between
                                       processors
```

```
        */UBYTE  jl—pad0;
];

/* JanusNode — an RPTR/Exec style node.
 */
struct JanusReqList
        [RPTR    jn—Succ;
        RPTR     jn—Pred;
        RPTR     jn—Name;
        UWORD    jn—ReqIndex;     /* this' index into jb—
                                     CommRegs */
];

#endif undef

/* JanusBase — the master data table for the janus project. It is
   located
 * at the bottom of parameter memory.*/

struct JanusBase [
        UBYTE  jb—Lock;           /* lock byte between
                                     processors */
        UBYTE  jb—8088Go;
        struct   JanusMemHead     /* free mem pool for param
                 jb—ParamMem;       memory
        struct   JanusMemHead     /* free mem pool for buffer
                                     memory */
                 jb—BufferMem;
        RPTR   jb—Interrupts;     /* (UBYTE *) of request
                                     byte-pairs */
        RPTR   jb—Parameters;     /* array of ptrs to parameter
                                     areas */
        UWORD jb—NumInterrupts;   /* number of interrupts &
                                     parameters */];

/* constant to set to indicate a pending software interrupt
*/#define JSETINT      0x7f
```

memrw.i—parameter area definition for access to other processors mem


```
#ifndef  JANUS_MEMRW_H
#define JANUS_MEMRW_H

/*
** this is the parameter block for the JSERV_READPC and JSERV_
** READAMIGA services — read and/or write the other processors
   memory.
*/
```

```
struct MemReadWrite [
        UWORD  mrw_Command;/* see below for list of commands */
        UWORD  mrw_Count;     /* number of bytes to transfer */
        ULONG  mrw_Address;   /* local address to access. This is */
                              /*   a machine pointer for the */
                              /*   68000, and a segment/offset */
                              /*   pair for the 808X. The ad- */
                              /*   dressed is arranged so the */
                              /*   native processor may read it */
                              /*   directly. */
        UWORD  mrw_Buffer;    /* The offset in buffer memory for */
                              /*   the other buffer. */
        UWORD  mrw_Status;    /* See below for status. */
];

/* command definitions */
#define MRWC_NOP          0     /* do nothing — return OK
                                   status code */
#define MRWC_READ         1     /* xfer from address to buffer */
#define MRWC_WRITE        2     /* xfer from buffer to address */
#define MRWC_READIO       3     /* only on 808x — read from
                                   IO space */
#define MRWC_WRITEIO      4     /* only on 808x — write to
                                   IO space */
#define MRWC_WRITEREAD 5        /* write from buffer, then read
                                   back */

/* status definitions */
#define MRWS_INPROGRESS$ffff  /* we've noticed cmd and are
                                 working on it */
#define MRWS_OK           $0000/* command completed OK */
#define MRWS_ACCESSERR    $0001/* some sort of protection
                                  violation */
#define MRWS_BADCMD       $0002/* command that the server
                                  doesn't understand */
```



```
#ifndef  JANUS_SERVICES_H
#define JANUS_SERVICES_H

/**/
/* this is the table of hard coded services. Other services may exist
/* that are dynamically allocated.
/**/
```

```
/* service numbers constrained by hardware */
#define JSERV_MINT      0   /* monochrome display written to */
#define JSERV_CINT      1   /* color display written to */
#define JSERV_CRT1INT   2   /* mono display's control registers
                               changed */
#define JSERV_CRT2INT   3   /* color display's control registers
                               changed */
#define JSERV_ENBKB     4   /* keyboard ready for next
                               character */
#define JSERV_LPT1INT   5   /* parallel control register */
#define JSERV_COM2INT   6   /* serial control register */

/* hard coded service numbers */
#define JSERV_PCBOOTED      7   /* PC is ready to service soft
                                   interrupts */
#define JSERV_SCROLL        8   /* PC is scrolling its screen */
#define JSERV_HARDDISK      9   /* Amiga reading PC hard disk */
#define JSERV_READAMIGA     10  /* PC reading Amiga mem */
#define JSERV_READPC        11  /* Amiga reading PC mem */
#define JSERV_AMIGACALL     12  /* PC causing Amiga function call */
#define JSERV_PCCALL        13  /* Amiga causing PC interrupt */
#define JSERV_NEWASERV      14  /* PC initiating Amiga side of a
                                   new service */
#define JSERV_NEWPCSERV     15  /* Amiga initiating PC side of a
                                   new service */

#endif  !JANUS_SERVICES_H
#ifndef JANUS_SETUPSIG_H#define
```



```
#ifndef EXEC_TYPES_H
#include "exec/types.h"
#endif

#ifndef EXEC_INTERRUPTS_H
#include "exec/interrupts.h"
#endif

struct SetupSig [
        struct Interrupt ss_Interrupt;
        APTR     ss_TaskPtr;
        ULONG    ss_SigMask;
        APTR     ss_ParamPtr;
        ULONG    ss_ParamSize;
        UWORD    ss_JanusIntNum;
];
#endif
```

PC JANUS SERVICE

This service is called via INT JANUS.
AH contains a function code

J_GET_SERVICE
Gets a new Service Number
Expects:
nothing
Returns:
AL : New Service Number to use
– 1 if no service available (J_NO_SERVICE)

J_GET_BASE
Gets Segments & offset of Janus Memory
Expects:
AL : Janus Service Number
Returns:
ES : Janus Parameter Segment
DI : Janus Parameter Offset (if defined),
else – 1
DX : Janus Buffer Segment
AL : Status (J_OK, J_NO_SERVICE)

J_ALLOC_MEM
Allocates Janus Memory
Expects:
AL : Type of memory to allocate
BX : Number of Bytes to allocate
Returns:
BX : Offset of registered memory if success,
AL : Status (J_OK, J_NO_MEMORY)

J_FREE_MEM
Releases Janus Memory
Expects:
AL : Type of memory to free
BX : Offset of Memory to free
Returns:
Crash if offset/type was wrong (J_GOODBYE, later)

J_SET_PARAM
Set the default parameter memory pointer
Expects:
AL : Janus Service Number to support
BX : Default Offset of Param Memory to install
Returns:
AL : Status (J_OK, J_NO_SERVICE)

J_SET_SERVICE
Set an address for a far call for that service
Expects:
AL : Janus Service Number to support
ES:DX: Entry address for FAR call
Returns:
AL : Status (J_OK, J_NO_SERVICE)

J_STOP_SERVICE
Prevents AMIGA from using the far call (see above) for this function and releases this Service Number.
No memory is freed up.
No calls are accepted from either side anymore.
Expects:
AL : Number of Service to stop
Returns:
AL: Status (J_OK, J_NO_SERVICE)

J_CALL_AMIGA
Calls the requested function on AMIGA side.
Does not wait for the call to complete.
If J_SET_SERVICE defined, it is internally called on completion.
Expects:
AL : AMIGA Service to call
BX : New Parameter Memory offset to use, −1 : Use default offset
Returns:
AL: Status (J_PENDING, J_FINISHED, J_NO_SERVICE)

J_WAIT_AMIGA
Waits for a previous issued J_CALL_AMIGA to complete.
This function is used if no J_SET_SERVICE is defined.
Expects:
AL : Service Number to wait for
Returns:
AL : Status (J_FINISHED, J_NO_SERVICE)

J_CHECK_AMIGA
Checks completion status of a pending J_CALL_AMIGA
Expects:
AL : Service Number to check
Returns:
AL : Status (J_PENDING, J_FINISHED, J_NO_SERVICE)

This is the Interrupt we are using:

```
JANUS           equ     Obh
```

These are the function codes we know:

```
J_GET_SERVICE   equ     0
J_GET_BASE      equ     1
J_ALLOC_MEM     equ     2
```

```
J_FREE_MEM      equ     3
J_SET_PARAM     equ     4
J_SET_SERVICE   equ     5
J_STOP_SERVICE  equ     6
J_CALL_AMIGA    equ     7
J_WAIT_AMIGA    equ     8
J_CHECK_AMIGA   equ     9
Status Returns:
J_NO_SERVICE    equ     0ffh    ; no service available
J_PENDING       equ     0       ; after J_CALL_AMIGA and
                                  J_CHECK_AMIGA
J_FINISHED      equ     1       ; after J_CALL_AMIGA and
                                  J_CHECK_AMIGA
J_OK            equ     0       ; general good return
J_NO_MEMORY     equ     3       ; requested memory not available
J_ILL_FNCTN     equ     4       ; Illegal function code used in AH
```

Disk request structure for higher level Amiga file request from 8086:

```
AmigaDskReq     STRUC
adr_Fnctn       DW      ?       function code (see below)
adr_File        DW      ?       file number
adr_Offset_h    DW      ?       byte offset into file high
adr_Offset_l    DW      ?       byte offset into file low
adr_Count_h     DW      ?       number of bytes to transfer high
adr_Count_l     DW      ?       number of bytes to transfer low
adr_BufferAddr  DW      ?       offset into MEMF_BUFFER memory
                                  for buffer
adr_Err         DW      ?       return code, 0 if all OK
AmigaDskReq     ENDS
```

Function codes for AmigaDskReq adr_Fnctn word

```
ADR_FNCTN_INIT          EQU     0       currently not used
ADR_FNCTN_READ          EQU     1       given file, offset, count, buffer
ADR_FNCTN_WRITE         EQU     2       given file, offset, count, buffer
ADR_FNCTN_SEEK          EQU     3       given file, offset
ADR_FNCTN_INFO          EQU     4       currently not used
ADR_FNCTN_OPEN_OLD      EQU     5       given ASCIIZ pathname in buffer
ADR_FNCTN_OPEN_NEW      EQU     6       given ASCIIZ pathname in buffer
ADR_FNCTN_CLOSE         EQU     7       given file
ADR_FNCTN_DELETE        EQU     8       given ASCIIZ pathname in buffer
```

Error codes for adr_Err, returned in low byte

```
ADR_ERR_OK              EQU     0       no error
ADR_ERR_OFFSET          EQU     1       not used
ADR_ERR_COUNT           EQU     2       not used
ADR_ERR_FILE            EQU     3       file does not exist
ADR_ERR_FNCT            EQU     4       illegal function code
ADR_ERR_EOF             EQU     5       offset past end of file
ADR_ERR_MULPL           EQU     6       not used
ADR_ERR_FILE_COUNT      EQU     7       too many open files
ADR_ERR_SEEK            EQU     8       seek error
ADR_ERR_READ            EQU     9       read went wrong
ADR_ERR_WRITE           EQU     10      write error
ADR_ERR_LOCKED          EQU     11      file is locked
```