INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION

INTERNATIONAL CENTRE FOR SCIENCE AND HIGH TECHNOLOGY
c/o INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS 34100 TRIESTE (ITALY) VIA GRIGNANO, 9 (ADRIATICO PALACE) P.O. BOX 586 TELEPHONE 040-224572 TELEFAX 040-224575 TELEX 460449 APH I

SMR/474 - 18

COLLEGE ON
"THE DESIGN OF REAL-TIME CONTROL SYSTEMS"
1 - 26 October

EXTENSIONS TO OS-9

H. VON DER SCHMITT
C.E.R.N.
PPE Division
CH-1211 Geneva 23
Switzerland

These are preliminary lecture notes, intended only for distribution to participants.

Extensions to OS-9

H. von der Schmitt

Real-Time College, Trieste, 1-27 October 1990

OS-9 has been designed originally for the Motorola 6809 microprocessor, around the year 1980. It is a complete, real-time, operating system (as opposed to real-time kernels) modeled after the UNIX operating system.

As compared to UNIX, OS-9 was targetet for "small" systems. For example, UNIX cannot run on a Rosy station because it requires very much memory, fast CPU, and fast large disk storage. There are today microprocessor systems, often based on the Motorola 68k family of processors, which have enough memory, CPU speed, and disk performance to run UNIX or OS-9.

As a kind of world standard for operating systems, UNIX offers of course much more existing software (e.g. wide choice of compilers, network software) than OS-9. Still, OS-9 is preferred for many applications: First of all, it is designed as a real-time system. Second, it is based on the concept of "memory modules" which goes together very well with the UNIX concept of "pipes" and "filters" to support modern software techniques of encapsulation and re-usability of code. It keeps the system small and easily maintainable.

OS-9 covers a very wide range of system performance. Starting with the 6809, it is now most widely used on the 68k family of 16/32 bit processors (68000 / 68010, 68020 / 68030, 68040 / more to come) and is as OS-9000 also available as a portable operating system (like UNIX) for different processor types. The higher performance hardware base allows also the construction of distributed computer systems by networking.

Many 68k systems manufactured by industry, e.g. for sensors, image processing, factory automation, are based on OS-9. Also, many small and big experiments in nuclear and high-energy physics, which incorporate many hundred microprocessors, use OS-9.

These lectures show extra features and differences of OS9/68k and OS9000 as compared to OS-9 for the 6809. The hardware base of the 68k family is explained briefly. An example of distributed processing with several CPUs running OS-9 is given.

EXTENSIONS TO OS-9

OS-9: originally made for 6809.

Today very widely used:
Motorola 68k family of processors.

Quite different hardware, higher speed...

Still OS-9 /68K !

Nice: hardware details hidden to user
by operating system (OS-9)
and programming language (C).

However: real-time applications are
more hardware dependent
(speed, interrupts, bus features...),
assembler programming also.

Outline of the lectures:

Look at differences / extensions of
OS-9 /68K (for 68k family)
OS-9000 (portable to various CPU typ

as compared to OS-9 / 6809.

Look also at the changed hardware basis,
in particular 68k family.
Determines the evolution of OS-9, and
its range of applications.

Very wide range of applications:
small industrial measurement systems
factory automation
physics experiments, up to the largest one:
Will use distributed computing in a
multiprocessor system as an example.

Recall features of OS-9:

1) Modularity

all components of the running system are modules:

- the kernel with its auxiliary modules
- user code and data modules
- file managers, drivers, device descriptors.

Modules are linked to dynamically, using their name.

2) Position independence and sharability

program code references are always relative to the program counter, data references relative to pointer registers set up at run time.

- code is reentrable → sharable
- code modules can be of general use.

3) Unified input/output system

OS-9 uses a four layer I/O system (IOman, file manager, driver, descriptor) to hide the I/O device's specific features.

- I/O to disk, terminal, pipe... do not differ in principle
- a program's standard I/O paths can be simply re-routed, e.g. to pipes (→ filter processes) or command files
- interprocess communication is contained in the I/O system.

Hardware features required so far:

pc - relative and

pointer - relative addressing.

O.K. for 6809 with branch relative and indexed addressing. Similar on 68K.

Modular structure together with easy communication with and among the modules support the <u>modern</u> programming concepts "encapsulation" and "re-usability" of resources.

Effect: small, easy to configure and use.

Encapsulation :

The inner working of a code module remains hidden. Only its input and output are visible and defined.

Re-usability :

"Filters" e.g. for sorting, mathematical transformations etc. can be used in many different contexts.

→ a step towards the more general principle of object oriented programming.

Recall more features:

4) Of course, it is a real-time system
- deterministic reaction times to external stimuli (interrupts), according to priority.
- ease of writing drivers, connecting processes to interrupts.

5) It is a complete operating system
- file system
- shell
- editor, compiler, assembler, linker, debugger
- optional memory management hardware
- has time-slicing: multitasking, multiuser.

6) Process synchronisation, signals, intercept routines.

All of these features are also found
in OS-9/68K and OS-9000.
(for 68k family) (portable)

Plus some more:

- much enhanced utilities, e.g. shell
- extra system calls to "events" for mutual exclusion and signaling
- named pipes: /pipe/xyz
- more standard file managers available incl. SBF (tape), NFM (network)
- standard (TCP/IP) networking + OS9Net
- distinction between system state and user state → protection*

Last but not least: much CPU power*
large address space*
more flexible interrupts*.

*) these depend on new hardware.

Before continuing with OS-9,
a look at the hardware base:

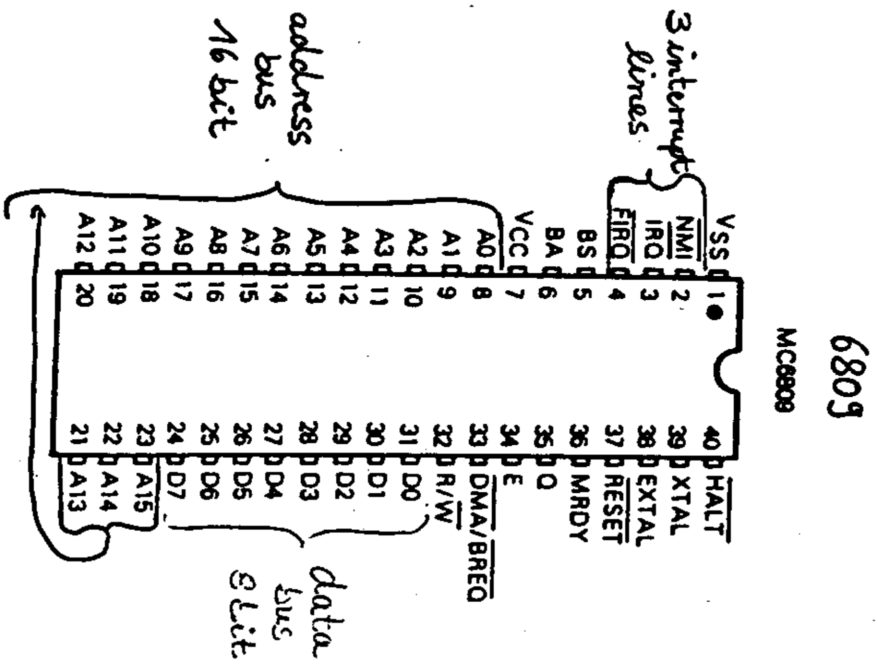
68 k processor family

- successor of 6809, for 32-bit applications. similar ideas for language and system support as 6809, but evolved further.
- very widespread use - "everywhere outside IBM PC": Atari ST, Macintosh, Hewlett-Packard, Apollo, ... most VMEbus modules.

→
- block diagram
- register set
- addressing modes } very brief
- instruction set }
- where does the speed come from: pipeline, cache, coprocessors

Comparison of processor features

Processor type	Data size (bits)	Address size (bits)	Address range (Mbyte)	Speed integer (relative)	Speed float (relative)	Price of chip (US$)	Number of transistors	typical clock rate (MHz)
6809	8/16	16	.064					2
68000/010	16/32	24	16	1	1	2	70.000	10
68020/030 with 68882	32	32	4096	6	60			20
68040	32	32	4096	20	450	600	1.200.000	33



68030

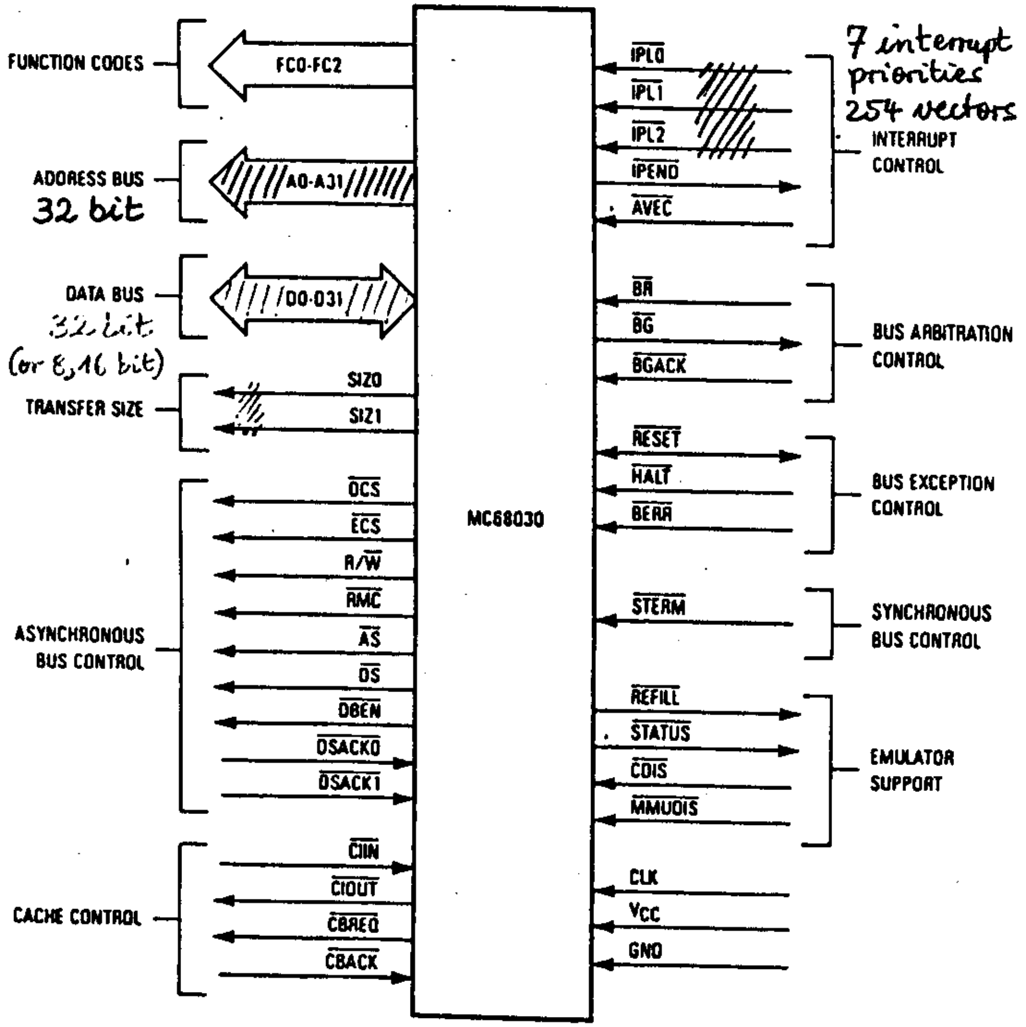


Figure 5-1. Functional Signal Groups

6809

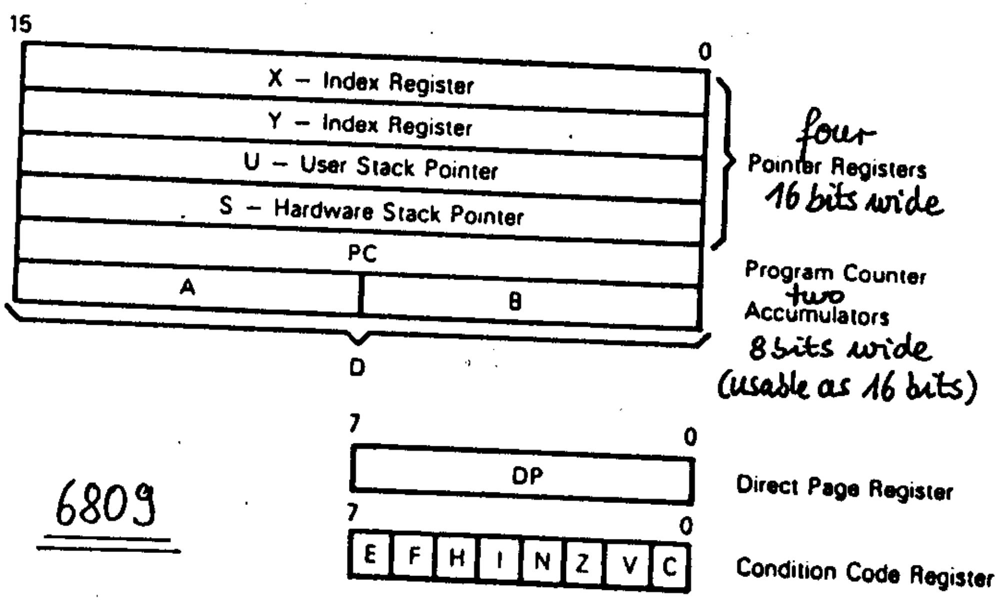
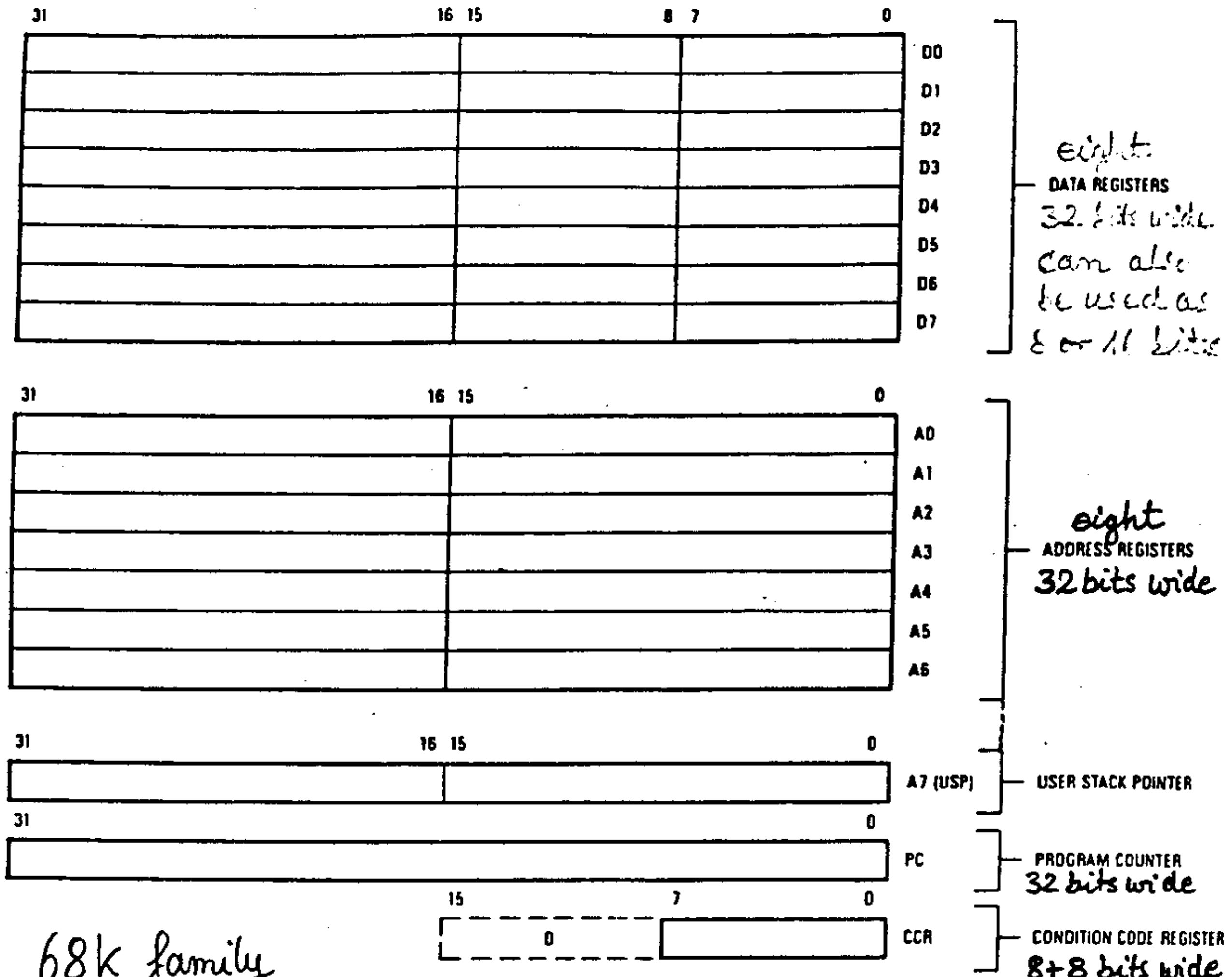


Figure 1-1. Programming Model



68k family

Figure 1-2. User Programming Model

11a

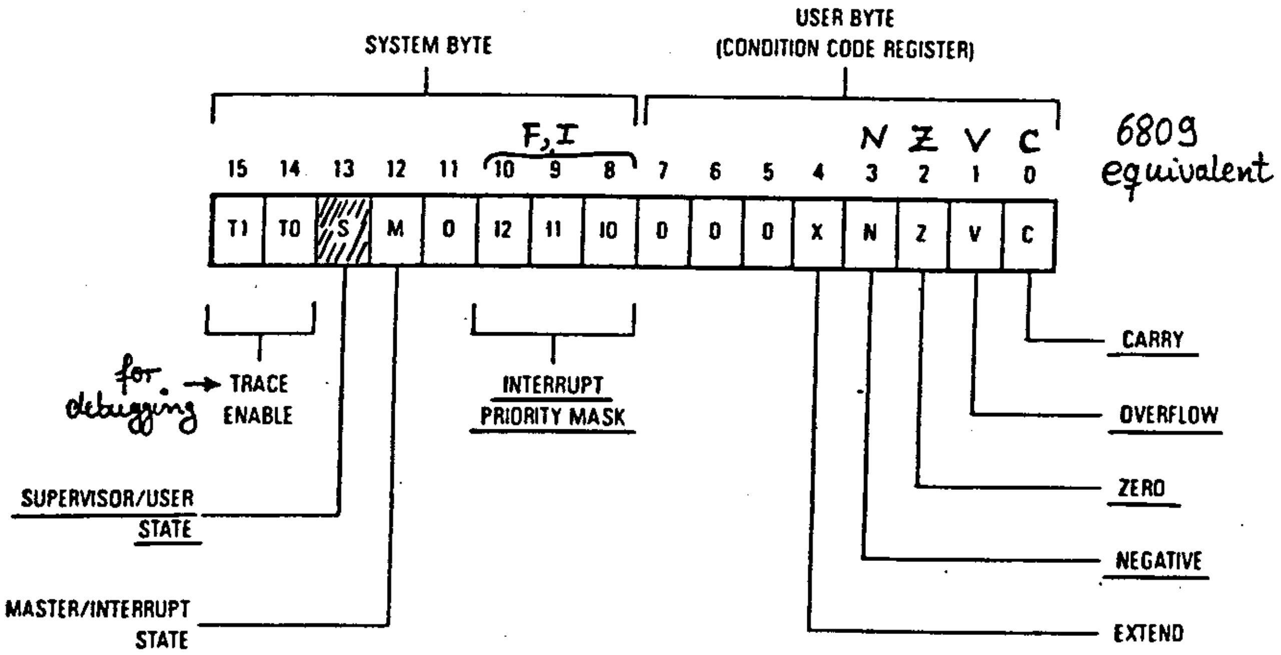


Figure 1-4. Status Register

68030

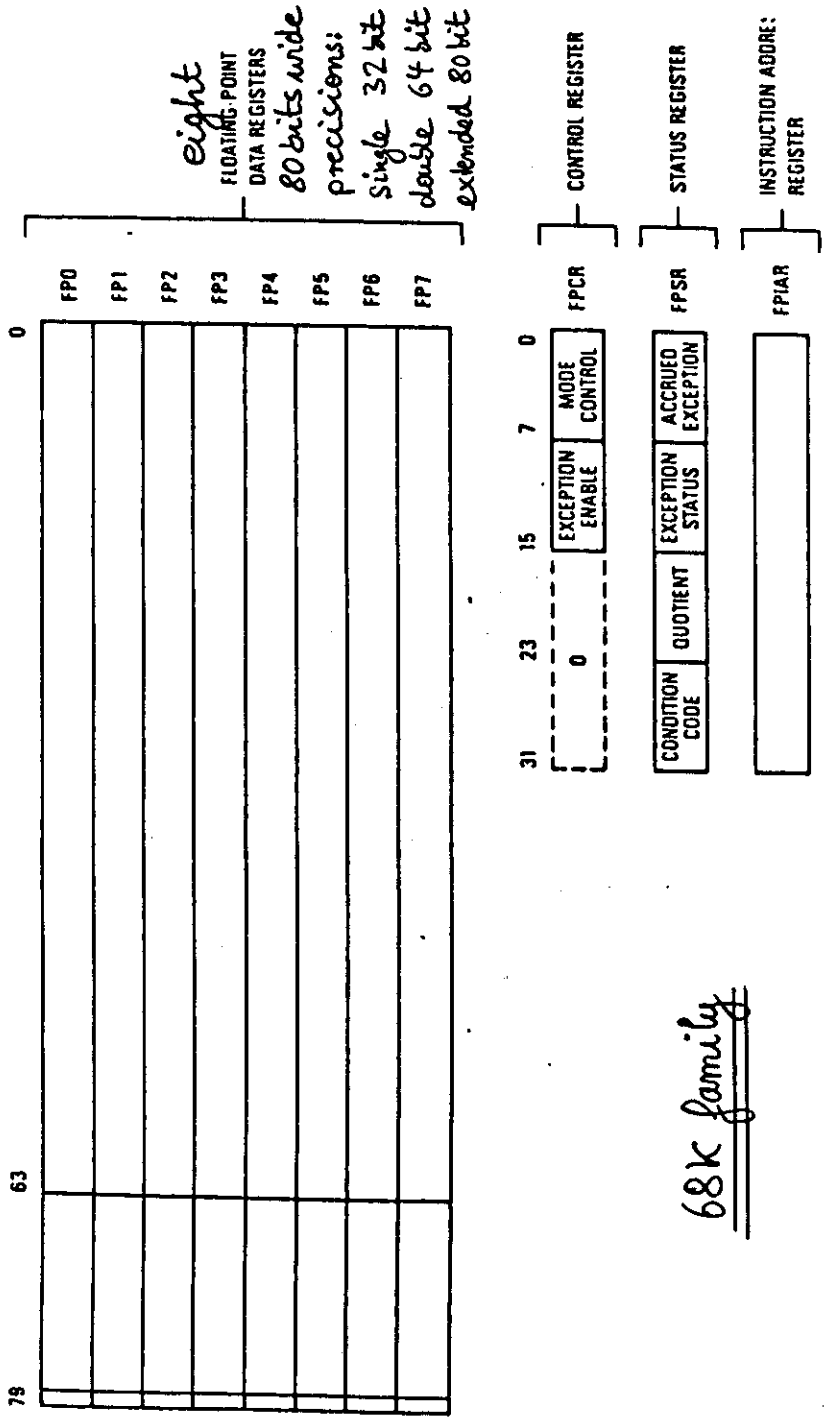


Figure 2-1. MC68881/MC68882 Programming Model

68K family

Addressing modes and instruction set

comparison 6809 / 68K

6809	68K
6809 is a one-address processor. Second operand defined in the operation code.	68K is a two-address processor. Both operands can be defined explicitely.
Operand size defined in the operation code.	Operand size can be chosen independently.

```
                        size →     ↓ source
                                     destination
LDA    #5               MOVE.B    #5,Dφ
LDD    #1000            MOVE.W    #1000,D1
LDX    #1000            MOVE.L    #1000,A6

CLR    16,X             CLR.B     (16,A6)
                        MOVE.L    (12,A6),(16,A6)

STD    --Y              MOVE.W    D1,-(A7)

MUL                     MULU.W    Dφ,D1

ASR    A,X              ASR.B     #1,(A6,Dφ)

INC    [2,X]            ADD.B     #1,([-2,A6])
```

The 68K has more registers:

8 data registers + 8 address registers.

The same register can be used as a

32-bit operand	≡	long	≡ .L
16	"	word	.W
8	"	byte	.B

These are too many combinations to be put into the operation code; thus the more "orthogonal" instructions of the 68K: decoupling of operation / size / 1st operand / 2nd operand.

Remarks:

- the operands cannot be chosen quite freely: 2nd operand can for most operations only be a register. Thus we call the 68K better a $1\frac{1}{2}$ address processor.
- the (almost) orhogonal instructions are easier to use - also compilers can generate better optimized code.
- address offsets for relative addressing can be 32 bits*, such that there are no restrictions arising from position independence and reentrance.

*) 68000/68010: only 16 bits.

68K family

Table 1-1. Addressing Modes

Addressing Modes	Syntax
Register Direct	
Data Register Direct	Dn
Address Register Direct	An
Register Indirect	
Address Register Indirect	(An)
Address Register Indirect with Postincrement	(An)+
Address Register Indirect with Predecrement	−(An)
Address Register Indirect with Displacement	(d_{16},An)
Register Indirect with Index	
Address Register Indirect with Index (8-Bit Displacement)	(d_8,An,Xn)
• Address Register Indirect with Index (Base Displacement)	(bd,An,Xn)
Memory Indirect	
• Memory Indirect Post-Indexed	([bd,An],Xn,od)
• Memory Indirect Pre-Indexed	([bd,An,Xn],od)
Program Counter Indirect with Displacement	(d_{16},PC)
Program Counter Indirect with Index	
PC Indirect with Index (8-Bit Displacement)	(d_8,PC,Xn)
• PC Indirect with Index (Base Displacement)	(bd,PC,Xn)
Program Counter Memory Indirect	
• PC Memory Indirect Post-Indexed	([bd,PC],Xn,od)
• PC Memory Indirect Pre-Indexed	([bd,PC,Xn],od)
Absolute	
Absolute Short	(xxx).W
Absolute Long	(xxx).L
Immediate	#⟨data⟩

•: not on 68000/010.

NOTES:

Dn = Data Register, D0-D7

An = Address Register, A0-A7

d_8, d_{16} = A twos-complement, or sign-extended displacement; added as part of the effective address calculation; size is 8 (d_8) or 16 (d_{16}) bits; when omitted, assemblers use a value of zero.

Xn = Address or data register used as an index register; form is Xn.SIZE*SCALE, where SIZE is .W or .L (indicates index register size) and SCALE is 1, 2, 4, or 8 (index register is multiplied by SCALE); use of SIZE and/or SCALE is optional.

bd = A twos-complement base displacement; when present, size can be 16 or 32 bits.

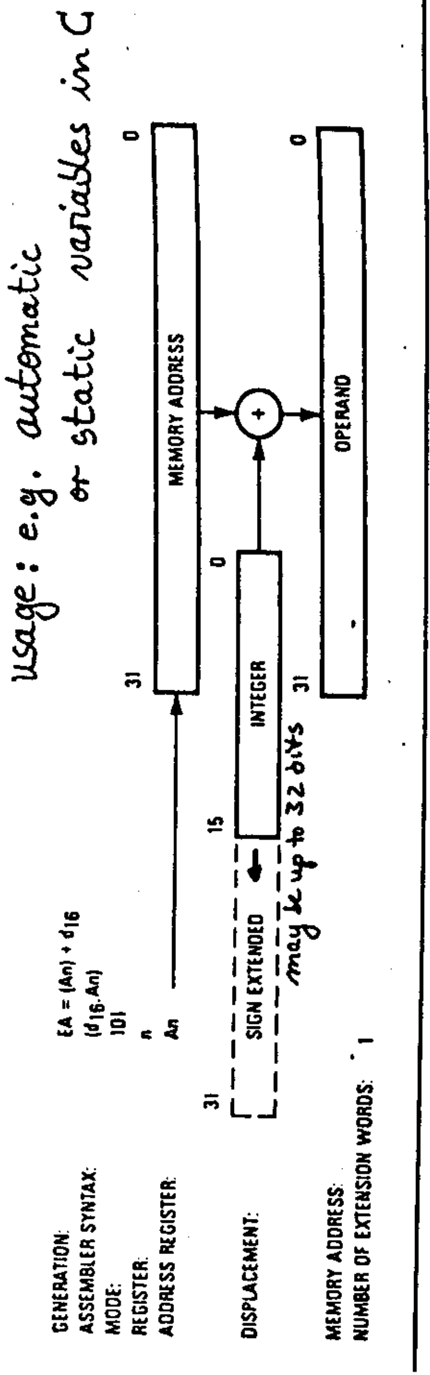
od = Outer displacement, added as part of effective address calculation after any memory indirection; use is optional with a size of 16 or 32 bits.

PC = Program Counter

⟨data⟩ = Immediate value of 8, 16, or 32 bits

( ) = Effective Address

[ ] = Use as indirect access to long word address



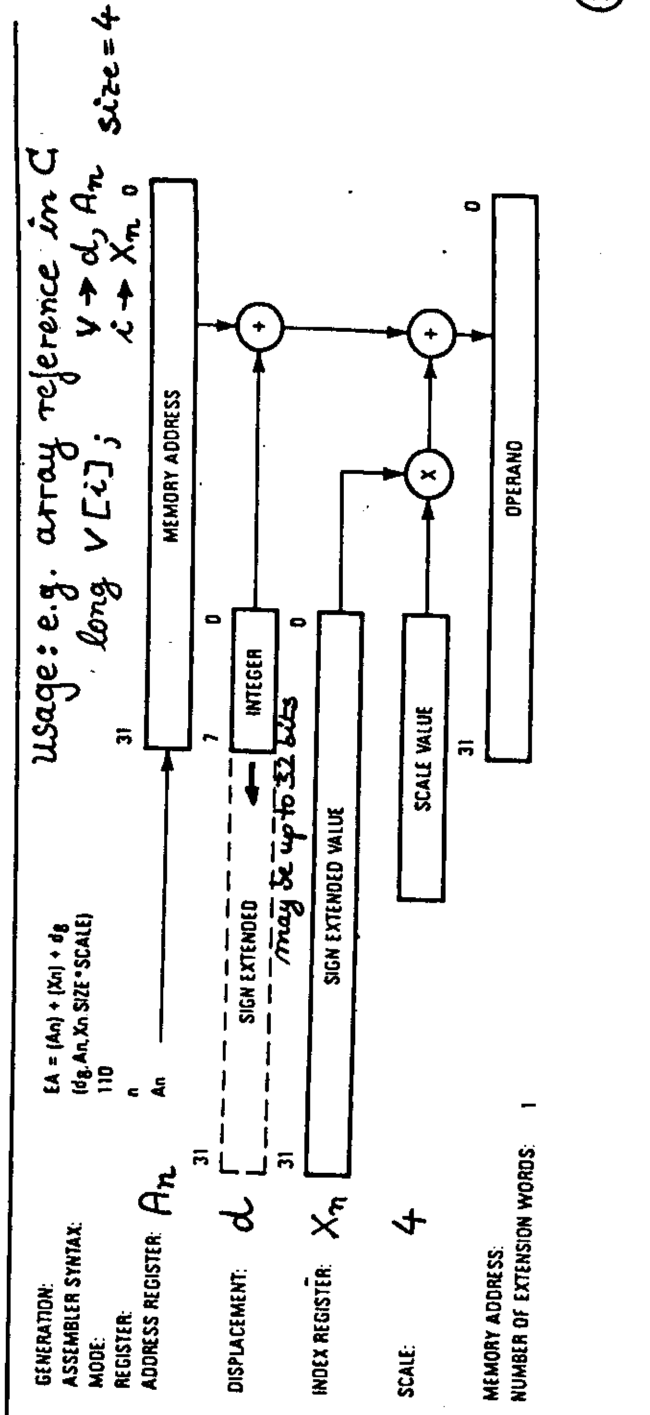


Table 1-2. Instruction Set

Mnemonic	Description
ABCD	Add Decimal with Extend
ADD	Add
ADDA	Add Address
ADDI	Add Immediate
ADDQ	Add Quick
ADDX	Add with Extend
AND	Logical AND
ANDI	Logical AND Immediate
ASL, ASR	Arithmetic Shift Left and Right
Bcc	Branch Conditionally
BCHG	Test Bit and Change
BCLR	Test Bit and Clear
BFCHG	Test Bit Field and Change
BFCLR	Test Bit Field and Clear
BFEXTS	Signed Bit Field Extract
BFEXTU	Unsigned Bit Field Extract
BFFFO	Bit Field Find First One
BFINS	Bit Field Insert
BFSET	Test Bit Field and Set
BFTST	Test Bit Field
BKPT	Breakpoint
BRA	Branch
BSET	Test Bit and Set
BSR	Branch to Subroutine
BTST	Test Bit
CAS	Compare and Swap Operands
CAS2	Compare and Swap Dual Operands
CHK	Check Register Against Bound
CHK2	Check Register Against Upper and Lower Bounds
CLR	Clear
CMP	Compare
CMPA	Compare Address
CMPI	Compare Immediate
CMPM	Compare Memory to Memory
CMP2	Compare Register Against Upper and Lower Bounds
DBcc	Test Condition, Decrement and Branch
DIVS, DIVSL	Signed Divide
DIVU, DIVUL	Unsigned Divide
EOR	Logical Exclusive OR
EORI	Logical Exclusive OR Immediate
EXG	Exchange Registers
EXT, EXTB	Sign Extend
ILLEGAL	Take Illegal Instruction Trap
JMP	Jump
JSR	Jump to Subroutine
LEA	Load Effective Address
LINK	Link and Allocate
LSL, LSR	Logical Shift Left and Right

Mnemonic	Description
MOVE	Move
MOVEA	Move Address
MOVE CCR	Move Condition Code Register
MOVE SR	Move Status Register
MOVE USP	Move User Stack Pointer
MOVEC	Move Control Register
MOVEM	Move Multiple Registers
MOVEP	Move Peripheral
MOVEQ	Move Quick
MOVES	Move Alternate Address Space
MULS	Signed Multiply
MULU	Unsigned Multiply
NBCD	Negate Decimal with Extend
NEG	Negate
NEGX	Negate with Extend
NOP	No Operation
NOT	Logical Complement
OR	Logical Inclusive OR
ORI	Logical Inclusive OR Immediate
PACK	Pack BCD
PEA	Push Effective Address
PFLUSH	Flush Entry(ies) in the ATC
PLOAD	Load Entry into the ATC
PMOVE	Move to/from MMU Registers
PTEST	Test a Logical Address
RESET	Reset External Devices
ROL, ROR	Rotate Left and Right
ROXL, ROXR	Rotate with Extend Left and Right
RTD	Return and Deallocate
RTE	Return from Exception
RTR	Return and Restore Codes
RTS	Return from Subroutine
SBCD	Subtract Decimal with Extend
Scc	Set Conditionally
STOP	Stop
SUB	Subtract
SUBA	Subtract Address
SUBI	Subtract Immediate
SUBQ	Subtract Quick
SUBX	Subtract with Extend
SWAP	Swap Register Words
TAS	Test Operand and Set
TRAP	Trap
TRAPcc	Trap Conditionally
TRAPV	Trap on Overflow
TST	Test Operand
UNLK	Unlink
UNPK	Unpack BCD

COPROCESSOR INSTRUCTIONS

cpBcc	Branch Conditionally
cpDBcc	Test Coprocessor Condition,

cpRESTORE	Restore Internal State of Coprocessor
cpSAVE	Save Internal State of Coprocessor

68K family

Table 3-2. Integer Arithmetic Operations

Instruction	Operand Syntax	Operand Size	Operation
ADD	Dn,(ea)	8, 16, 32	source + destination ♦ destination
	(ea),Dn	8, 16, 32	
ADDA	(ea),An	16, 32	
ADDI	#(data),(ea)	8, 16, 32	immediate data + destination ♦ destination
ADDQ	#(data),(ea)	8, 16, 32	
ADDX	Dn,Dn	8, 16, 32	source + destination + X ♦ destination
	−(An), −(An)	8, 16, 32	
CLR	(ea)	8, 16, 32	0 ♦ destination
CMP	(ea),Dn	8, 16, 32	destination − source
CMPA	(ea),An	16, 32	
CMPI	#(data),(ea)	8, 16, 32	destination − immediate data
CMPM	(An) + ,(An) +	8, 16, 32	destination − source
CMP2	(ea),Rn	8, 16, 32	lower bound (= Rn (= upper bound
DIVS/DIVU	(ea),Dn	32/16 ♦ 16:16	destination/source ♦ destination (signed or unsigned)
	(ea),Dr:Dq	64/32 ♦ 32:32	
	(ea),Dq	32/32 ♦ 32	
DIVSL/DIVUL	(ea),Dr:Dq	32/32 ♦ 32:32	
EXT	Dn	8 ♦ 16	sign extended destination ♦ destination
	Dn	16 ♦ 32	
EXTB	Dn	8 ♦ 32	
MULS/MULU	(ea),Dn	16 × 16 ♦ 32	source*destination ♦ destination (signed or unsigned)
	(ea),Dl	32 × 32 ♦ 32	
	(ea),Dh:Dl	32 × 32 ♦ 64	
NEG	(ea)	8, 16, 32	0 − destination ♦ destination
NEGX	(ea)	8, 16, 32	0 − destination − X ♦ destination
SUB	(ea),Dn	8, 16, 32	destination = source ♦ destination
	Dn,(ea)	8, 16, 32	
SUBA	(ea),An	16, 32	
SUBI	#(data),(ea)	8, 16, 32	destination − immediate data ♦ destination
SUBQ	#(data),(ea)	8, 16, 32	

Monadic Operation Format

Instruction	Operand Syntax	Operand Format	Operation
F(mop)	(ea),FPn FPm,FPn FPn	B,W,L,S,D,X,P X X	source ♦ function ♦ FPn FPn ♦ function ♦ FPn

where:
 <mop> is any one of the monadic operations specifiers.

68881 / 68882 Floating Point Coprocessor

Table 4-5. Monadic Operations

Instruction	Function
FABS	absolute value
FACOS	arc cosine
FASIN	arc sine
FATAN	arc tangent
FATANH	hyperbolic arc tangent
FCOS	cosine
FCOSH	hyperbolic cosine
FETOX	e^x
FETOXM1	$e^x - 1$
FGETEXP	extract exponent
FGETMAN	extract mantissa
FINT	extract integer part
FINTRZ	extract integer part, rounded-to-zero

Instruction	Function
FLOGN	ln(x)
FLOGNP1	ln(x + 1)
FLOG10	$\log_{10}(x)$
FLOG2	$\log_2(x)$
FNEG	negate
FSIN	sine
FSINH	hyperbolic sine
FSQRT	square root
FTAN	tangent
FTANH	hyperbolic tangent
FTENTOX	10^x
FTWOTOX	2^x

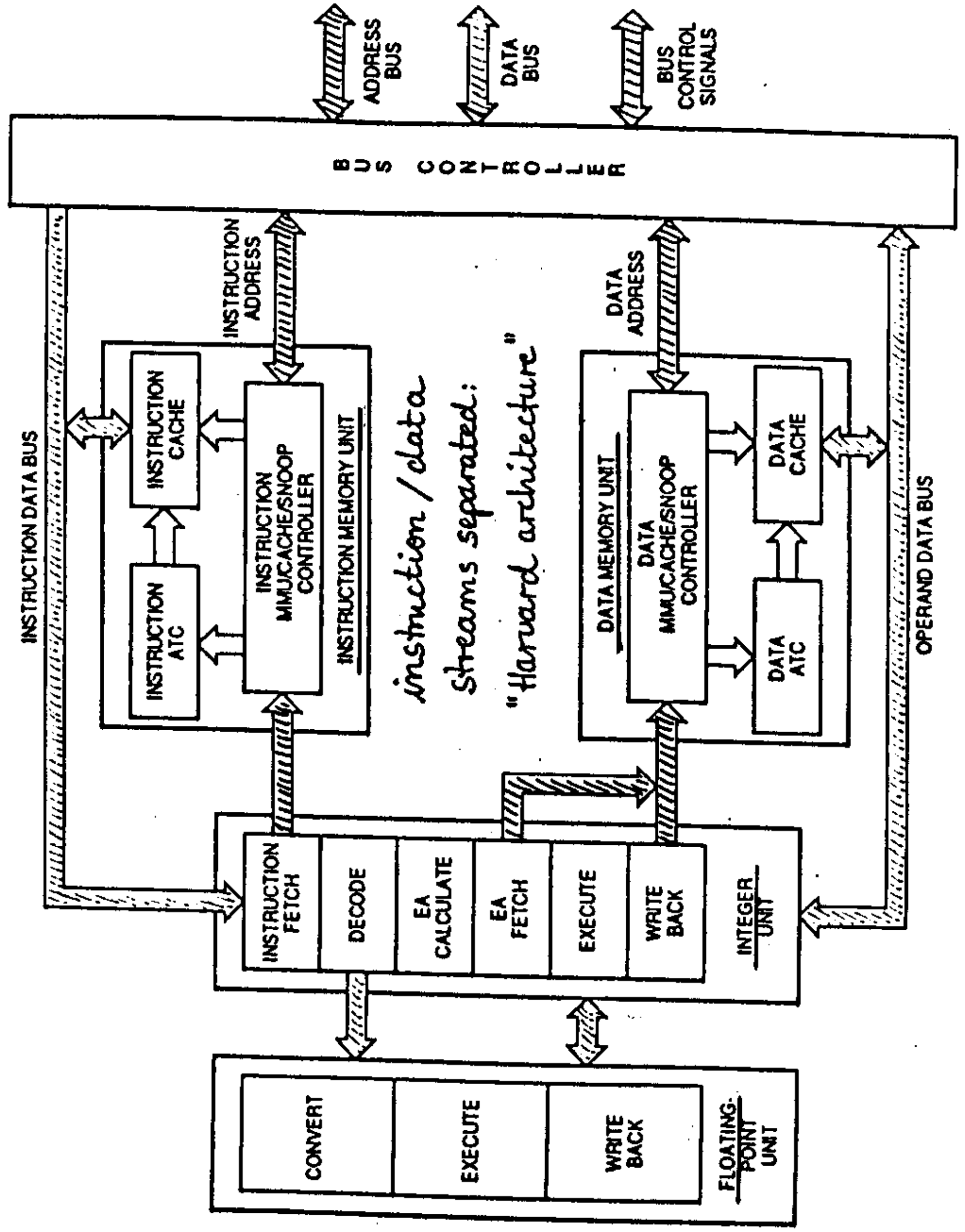
Exception Vector Assignments

Vector Number(s)	Vector Offset		Assignment	STATUS Asserted
	Hex	Space		
0	000	SP	Reset Initial Interrupt Stack Pointer	—
1	004	SP	Reset Initial Program Counter	—
2	008	SD	Bus Error	Yes
3	00C	SD	Address Error	Yes
4	010	SD	Illegal Instruction	No
5	014	SD	Zero Divide	No
6	018	SD	CHK, CHK2 Instruction	No
7	01C	SD	cpTRAPcc, TRAPcc, TRAPV Instructions	No
8	020	SD	Privilege Violation	No
9	024	SD	Trace	Yes
10	028	SD	Line 1010 Emulator	No
11	02C	SD	Line 1111 Emulator → program unchanged if no F.P. chip	No
12	030	SD	(Unassigned, Reserved)	
13	034	SD	Coprocessor Protocol Violation	No
14	038	SD	Format Error	No
15	03C	SD	Uninitialized Interrupt	Yes
16 Through 23	040 05C	SD SD	Unassigned, Reserved	
24	060	SD	Spurious Interrupt	Yes
25	064	SD	Level 1 Interrupt Autovector	Yes
26	068	SD	Level 2 Interrupt Autovector	Yes
27	06C	SD	Level 3 Interrupt Autovector	Yes
28	070	SD	Level 4 Interrupt Autovector	Yes
29	074	SD	Level 5 Interrupt Autovector Similar to 6809	Yes
30	078	SD	Level 6 Interrupt Autovector	Yes
31	07C	SD	Level 7 Interrupt Autovector	Yes
32 Through 47	080 0BC	SD SD	TRAP #0-15 Instruction Vectors	No No
48	0C0	SD	FPCP Branch or Set on Unordered Condition	No
49	0C4	SD	FPCP Inexact Result	No
50	0C8	SD	FPCP Divide by Zero	No
51	0CC	SD	FPCP Underflow	No
52	0D0	SD	FPCP Operand Error	No
53	0D4	SD	FPCP Overflow	No
54	0D8	SD	FPCP Signaling NAN	No
55	0DC	SD	Unassigned, Reserved	No
56	0E0	SD	MMU Configuration Error	No
57	0E4	SD	Defined for MC68851 not used by MC68030	
58	0E8	SD	Defined for MC68851 not used by MC68030	
59 Through 63	0EC 0FC	SD SD	Unassigned, Reserved	
64 Through 255	100 255	SD SD	User Defined Vectors (192)	

How interrupts work in 68K systems

There are 3 external interrupts to a 6809:
NMI, FIRQ, IRQ. They have fixed vectors:
$FFFC $FFF6 $FFF8.
If more than one device is connected to one interrupt input, the software has to find the interrupter by polling (OS-9 polling table).

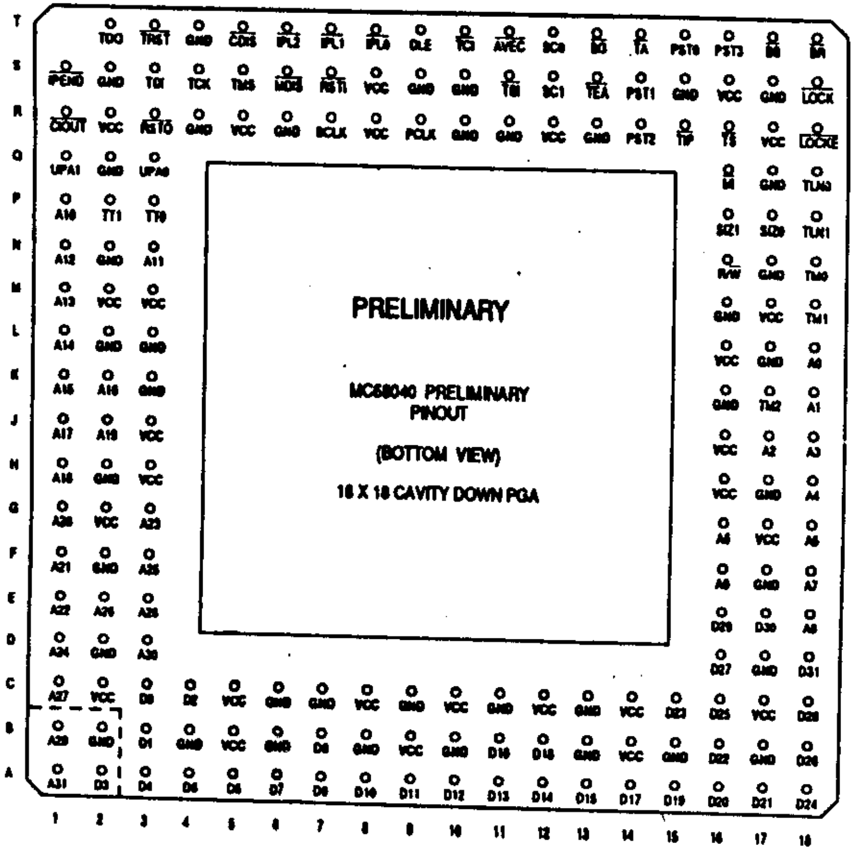
There are 7 external interrupt priorities, encoded into 3 bits IPLØ,1,2 to a 68K. The processor asks the interrupter to identify itself with a vector number (8 bit). This number selects one of 254 software routines directly (no polling necessary unless several devices use the same vector number).



instruction / data streams separated: "Harvard architecture"

12.2 PIN ASSIGNMENTS

The MC68040 is available in an 179-pin package. The following figure shows the pin assignment of the MC68040.



Example: 68040. As fast as many RISC CPUs.

Price for high performance:

very complex chip : 1.2 million transistors
complex board : chip has 179 pins
wants 128 bit

Back to OS-9 ...

Summary of the 68k hardware excursion:

- huge CPU performance range can be covered by OS-9
- virtually unlimited address space; many megabytes of memory usual
- 32-bit data size is standard
- efficient interrupt hardware.

We will see that OS-9/68K does not look very different from OS-9/6809... same concept, similar utilities, even almost same system calls. Many data structures used by OS-9 are however now based on 16 instead of 8 bits. (all module headers, contents of device descriptors, signal codes, etc.)

Of course, OS-9 had to be re-written completely for the 68k because it was (and is) written in assembler.

The new OS-9000 is now largely written in C for portability, at the cost of some memory and speed.

Some memory requirements of the three generations of OS-9:

	6809	68K	OS-9000
minumum useful RAM	40kb	200kb	300kb
kernel	3kb	27kb	56kb
shell	1kb	19kb	30kb

growing functionality →

Differences OS-9 6809 / 68k

1) the system:

- interprocess communication

  better handling of signals — F$Send

  new: events — F$Event

  named pipes — I$.... /pipe/xy

  data modules — F$DatM

  (events, named pipes, data modules: system global)

- more file managers for new device clas

  sequential block devices (magnetic tape) — SBF

  network devices: OS9Net (Ethernet, RS232, ...) — NFM

- system protection hardware

  68k "system state" plus "memory mapping unit" → optional system module — SSM

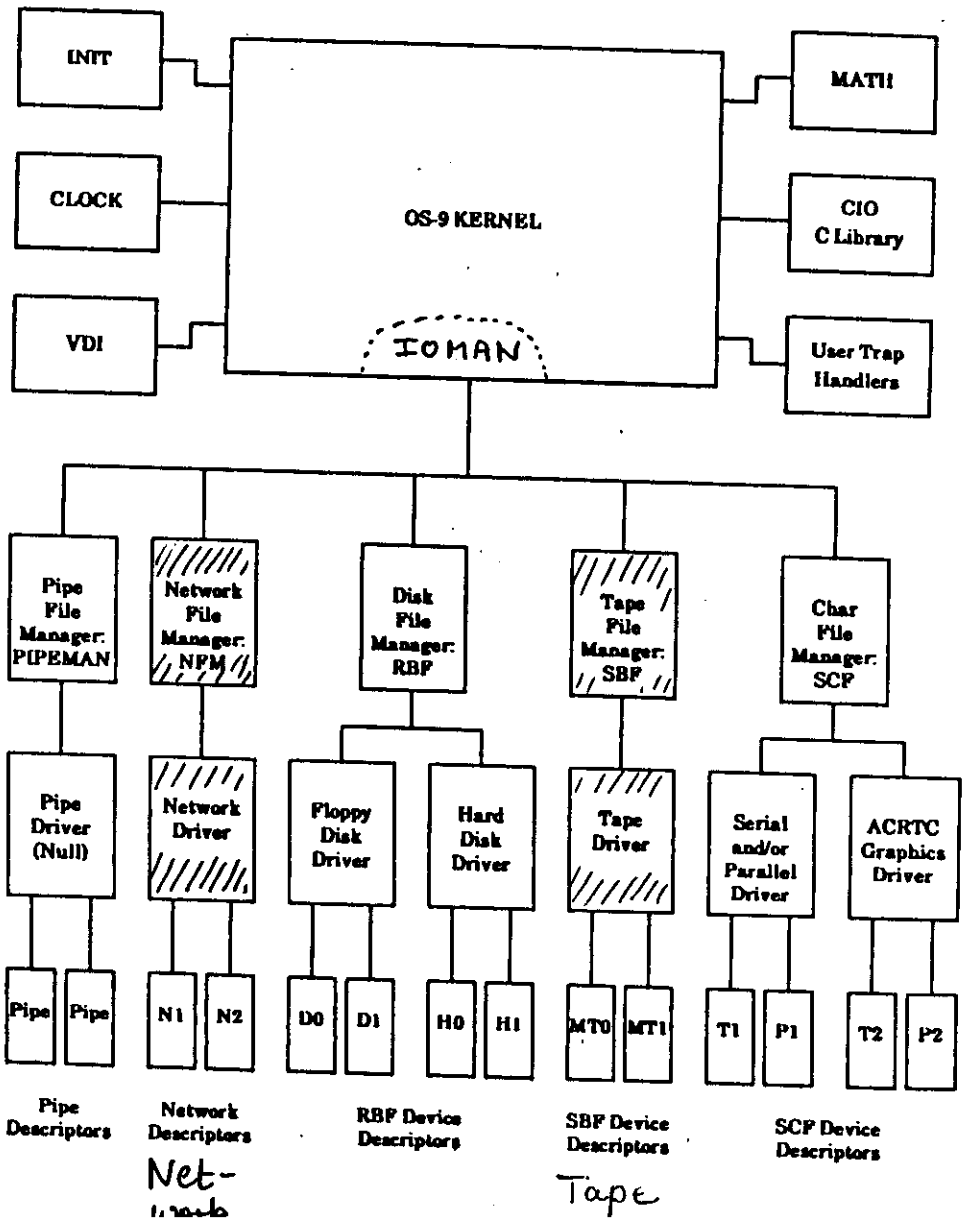
2) the utilities:

- enhanced "shell" command processor
  wildcards
  environment variables
- logical names embedded in system

The structure of OS-9 has remained unchanged!

OS-9 / 68K

Figure 1: OS-9 MODULE ORGANIZATION



OS9 /68K 6809

Offset	Usage		
$00	M$ID	Sync Bytes ($4AFC)	$00 - $01
$02	M$SysRev	Revision ID	%
$04	M$Size	Module Size	$02 - $03
$08	M$Owner	Owner ID	%
$0C	M$Name	Module Name Offset •	$04 - $05
$10	M$Accs	Access Permissions	%
$12	M$Type	Module Type	$06
$13	M$Lang	Module Language	
$14	M$Attr	Attributes	$07
$15	M$Revs	Revision Level	
$16	M$Edit	Edit Edition	%
$18	M$Usage	Usage Comments Offset •	%
$1C	M$Symbol	Symbol Table	%
$20	RESERVED		%
$2E	M$Parity	Header Parity Check	$08
$30-up	Module Type Dependent		
	Module Body		
	CRC Check		3 bytes

• These fields are offset to strings

Figure 3: Module Header Standard Fields

OS9 /68K

Offset	Usage		
$30	M$Port	Port Address	32 bit
$34	M$Vector	Trap Vector Number	2-255
$35	M$IRQLvl	IRQ Interrupt Level	1-7
$36	M$Prior	IRQ Polling Priority	0-255 *
$37	M$Mode	Device Mode Capabilities	
$38	M$FMgr	File Manager Name Offset	
$3A	M$PDev	Device Driver Name Offset	
$3C	M$DevCon	Device Configuration Offset	
$3E	Reserved		
$46	M$Opt	Initialization Table Size	
$48	M$DTyp	Device Type	

Figure 10: Additional Standard Header Fields For Device Descriptors

* because there are so many different interrupt vectors, polling is usually not necessary.

Interprocess communication

purposes: data exchange
synchronisation
mutual exclusion

- OS-9 signals

F$Send — to send a signal to a specified process.

F$Sleep — to await a signal.

F$Icpt — to install a routine for handling special signals.

Standard signal codes:

kill	
wakeup	used by driver's IRQ routine
interrupt	control - C on terminal
abort	control - E on terminal

- Signals carry a 16-bit information (signal code ; 6809: 8-bit).
- They are process specific, rather than system global (exception: ID = Ø).
- They are queued to the receiving process (were not in 6809). Exception: wakeup lost if process running!
- Processes can mask signals (not in 6809). Caution with real time response!

Signals are primarily useful for the internal working of I/O system. Careful when using explicitely for synchronisatio