

Astrophysikalisches Numerikum

part01: The Basics

Peter H. Hauschildt

yeti@hs.uni-hamburg.de

Hamburger Sternwarte
Gojenbergsweg 112
21029 Hamburg

Topics

- hardware essentials
- intro to unix
 - logging in
 - filesystems and files
 - useful commands
 - editors
- intro to make & compilers

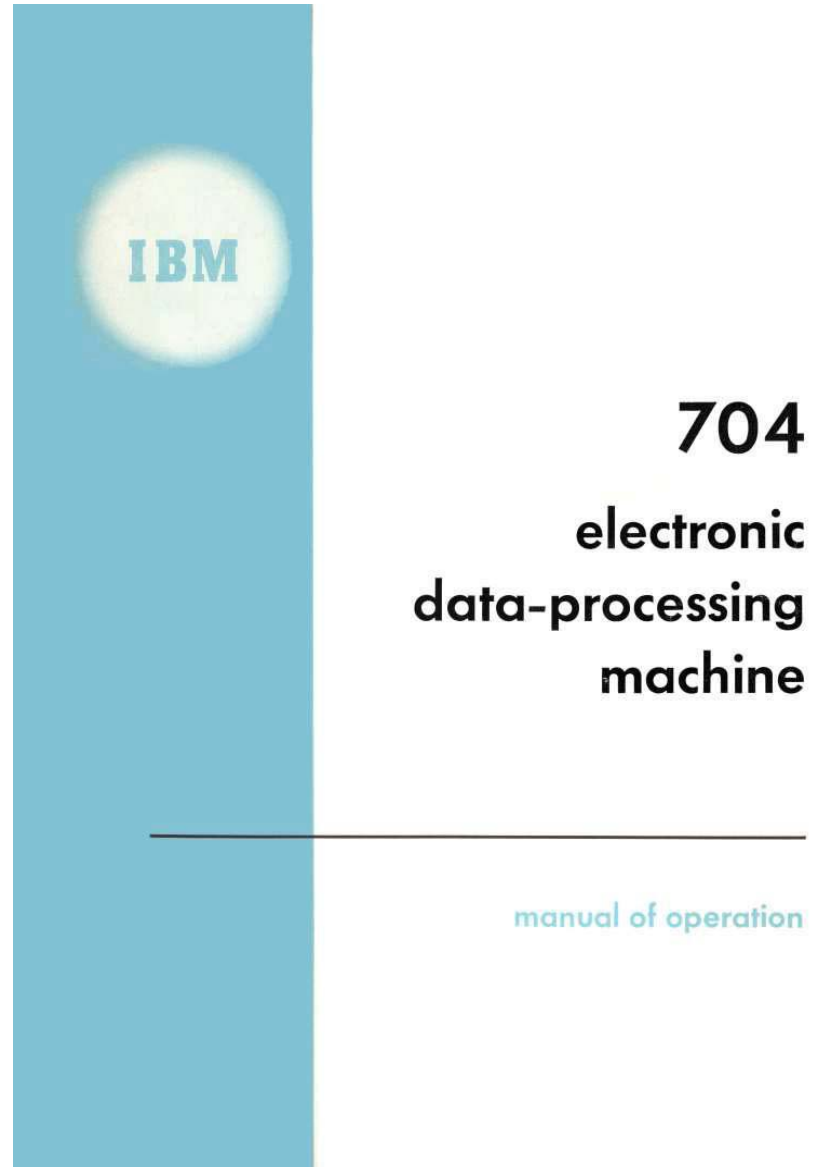
First Rule

Don't Panic! Don't Panic!
Don't Panic! Don't Panic!
Don't Panic!

hardware essentials

- computational astrophysics needs computers
- typical: problem size/complexity larger than available computers
- → need to get more out of a machine than designers intended in order to get the job done!
- → need to understand how computer works
- use this to squeeze everything and more out of it
- *know your tools!*

modern computers



Architecture of modern computers



Magnetic Core
Storage

Central
Processing
Unit

Magnetic Drum
Operator's Console

Power Supply
Printer
Card Reader

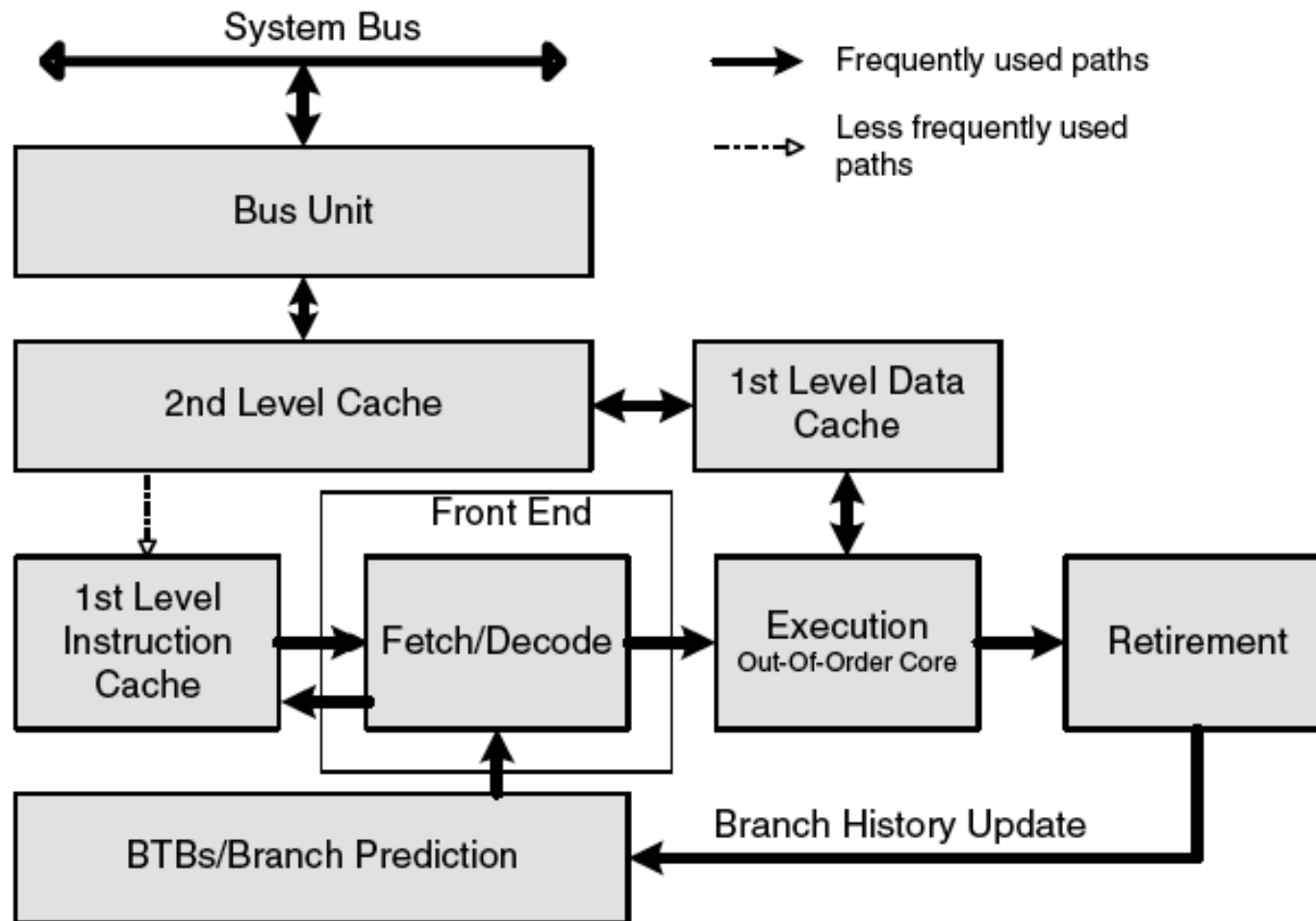
Card Punch

Magnetic Tape Units

IBM 704 ELECTRONIC DATA-PROCESSING MACHINES

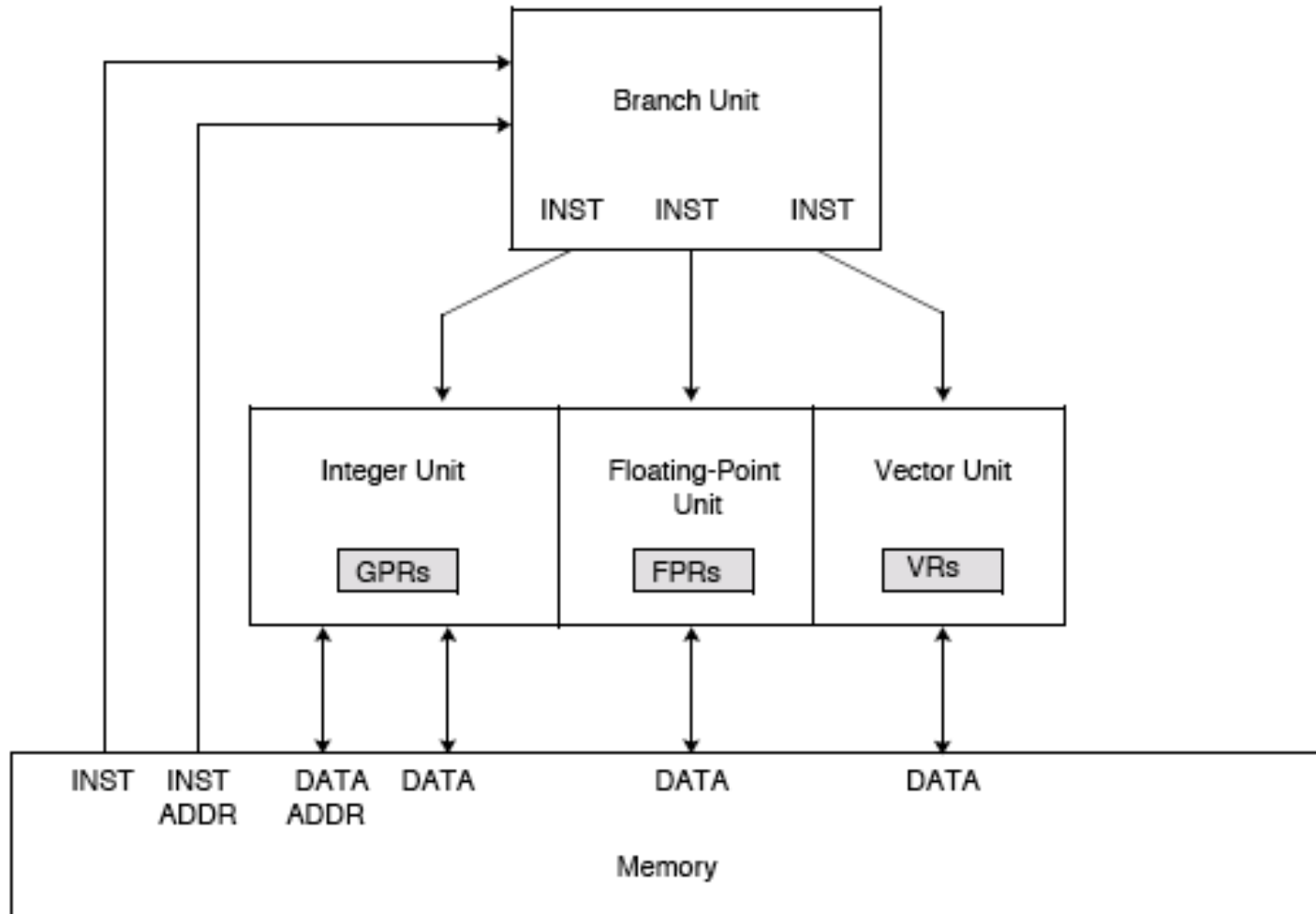
CPU architectures I

Figure 1-5 The Intel Pentium M Processor Microarchitecture

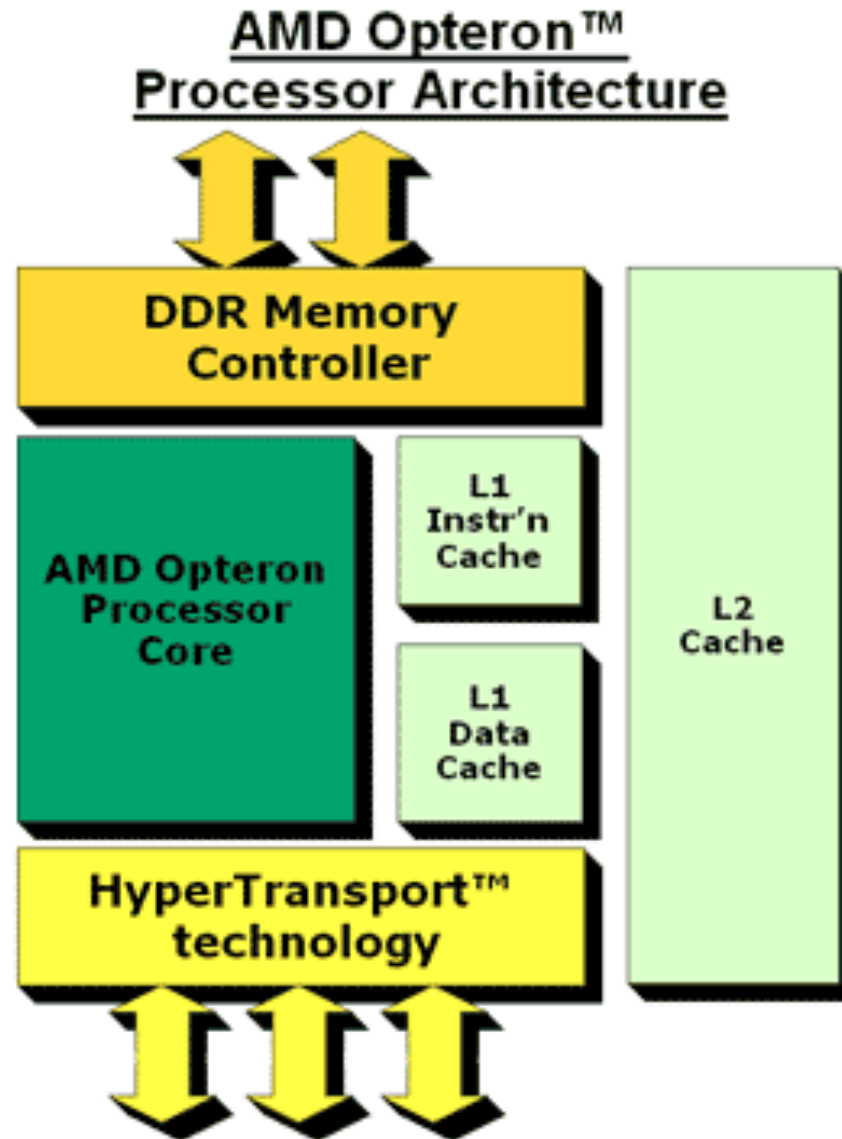


CPU architectures II

Figure 1-1. High Level Structural Overview of PowerPC with AltiVec Technology



CPU architectures III



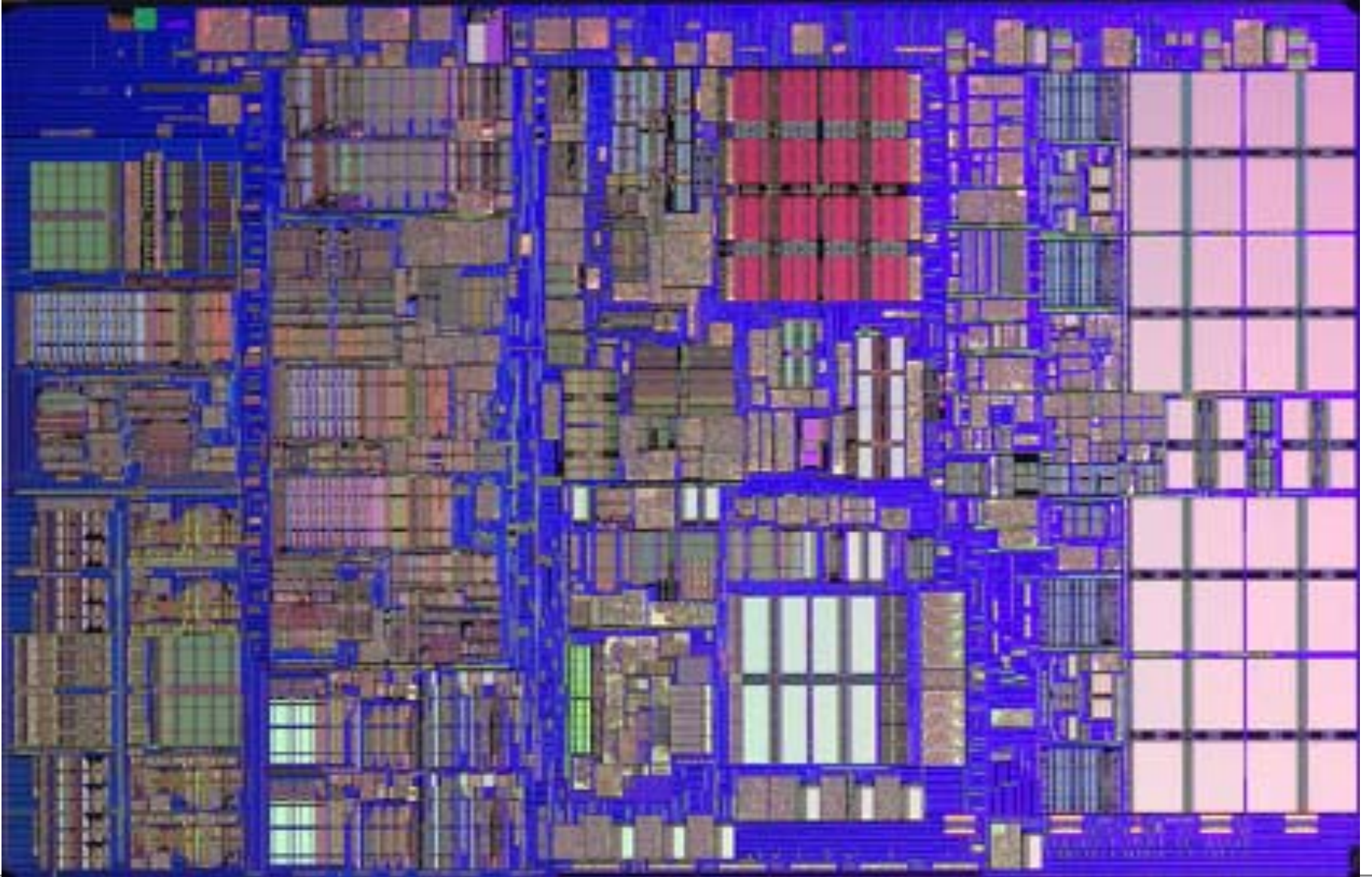
CPU features

- Memory controller:
 - translates virtual to hardware memory addresses
 - data transfer external memory $\leftrightarrow$ cache
 - prefetches data
- L1/L2/L3 data/instruction caches
 - on-chip
 - very fast (L1 fastest, L3 slowest)
 - very expensive (\$\$\$ & space!)

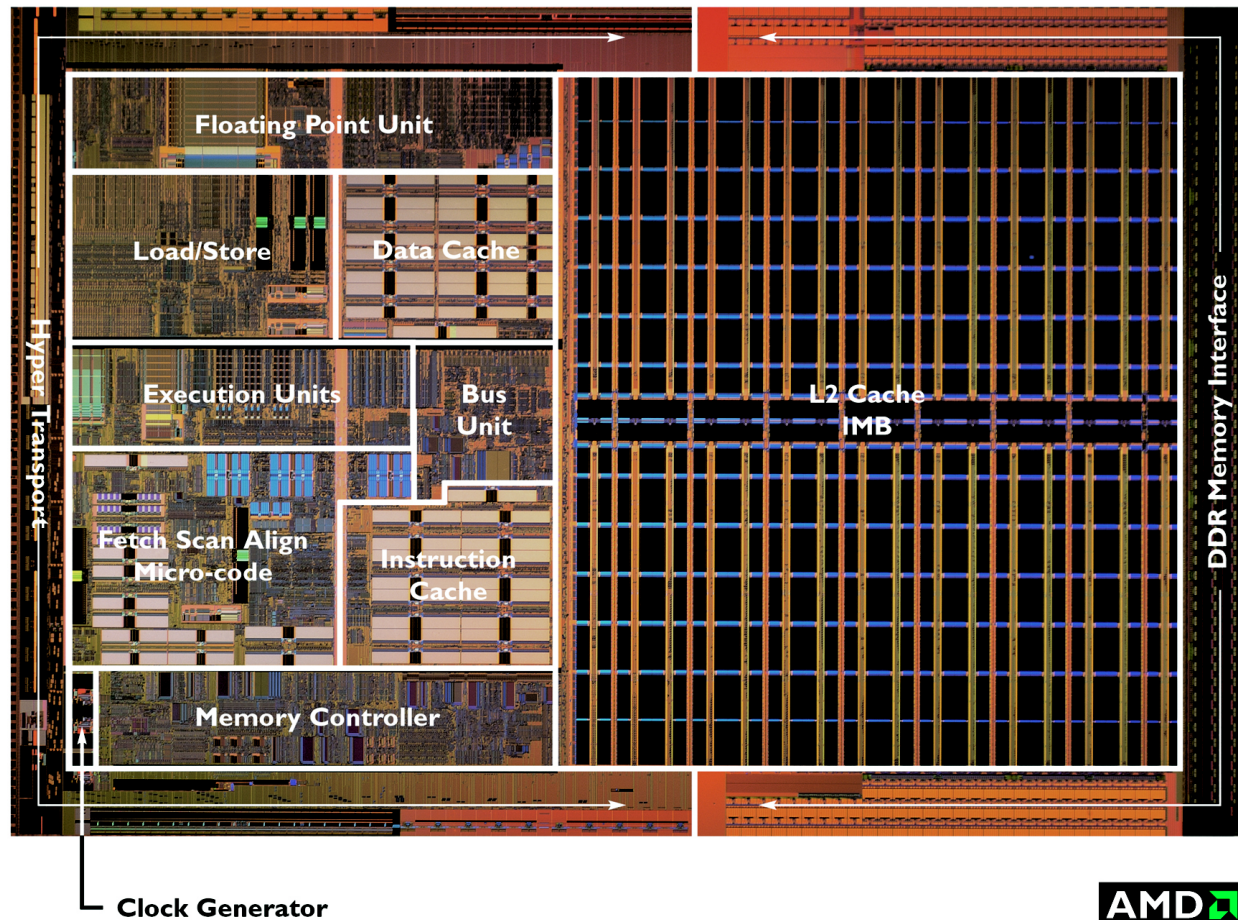
CPU features

- processor core
 - instruction fetch/decode/ordering/scheduler
 - branch unit(s)
 - 1 or more integer units
 - 2 or more floating point units (FPUs)
 - 0 or more vector units
- each unit includes data registers

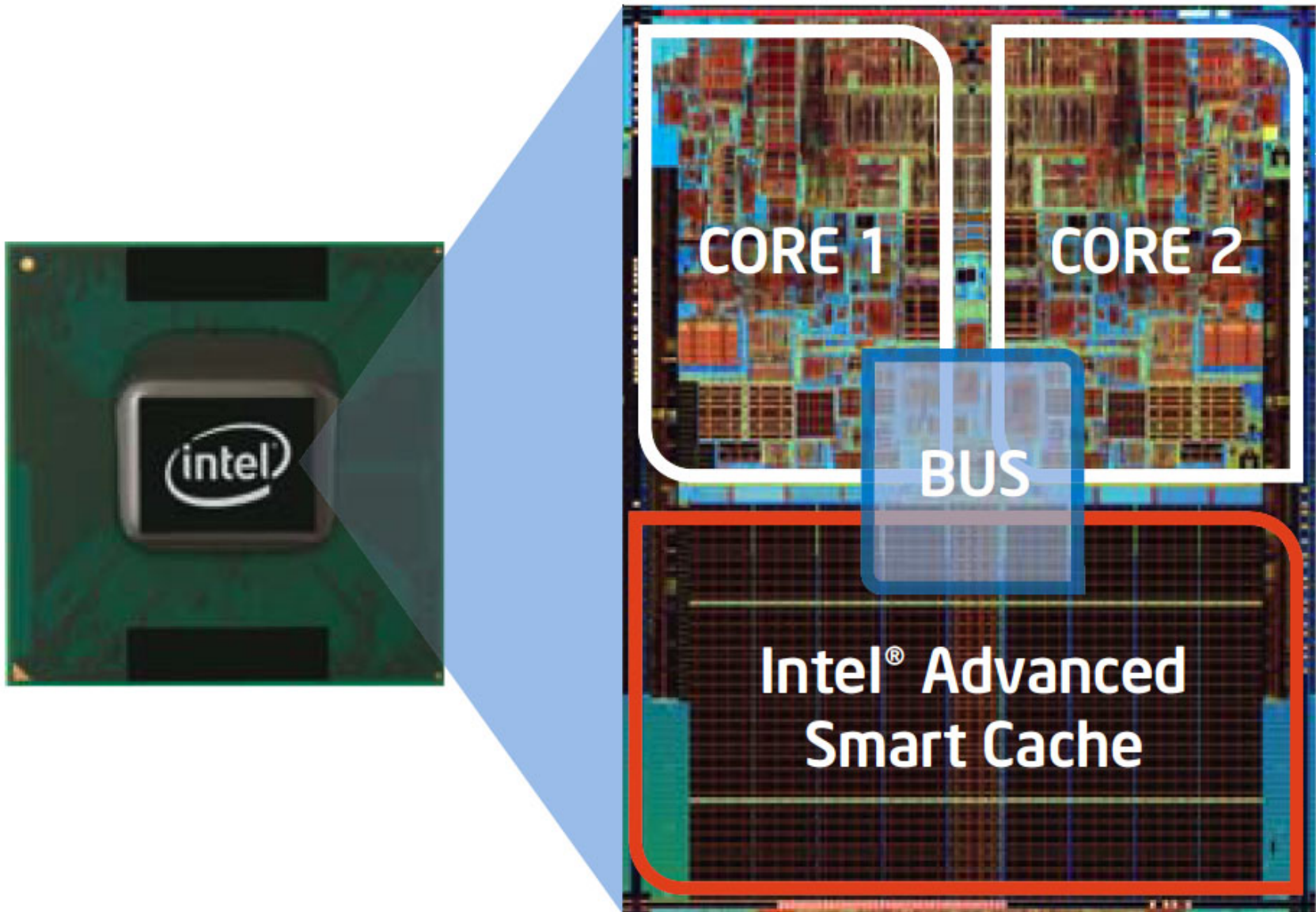
PPC970



Opteron



Core2 duo

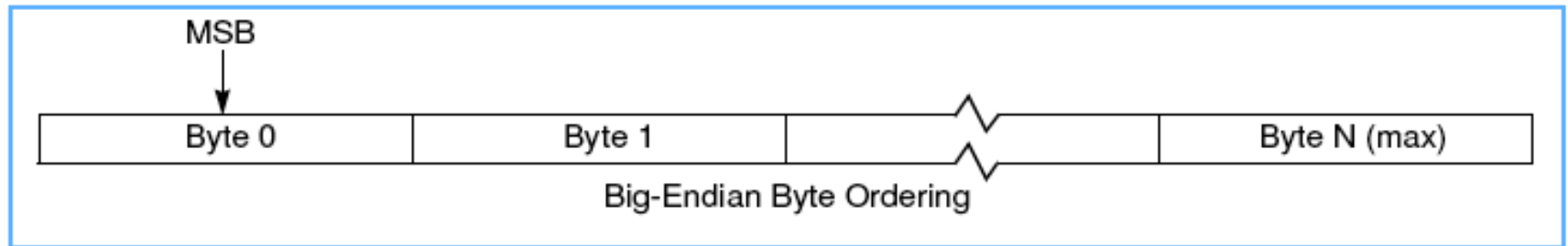


CPU features

- address space (32/64 bits)
 - limits available virtual address space
 - → max. size of data
- data storage in memory (Big/Little Endian)

data storage: Big Endian

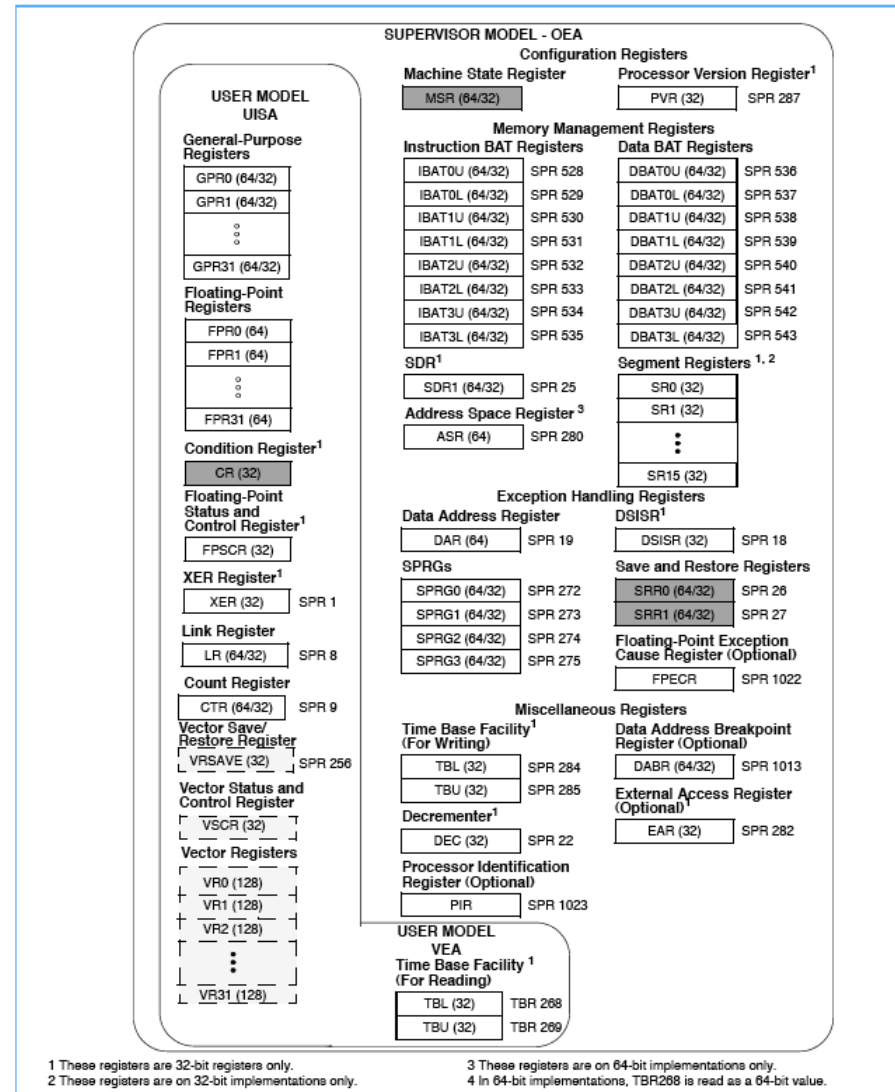
Figure 1-2. Big-Endian Byte and Bit Ordering



- important for binary data exchange!
- Big Endian: Power, PPC, MIPS, PA-RISC
- Little Endian: IA-32, IA-64, AMD64, Alpha

vector registers (PPC970)

Figure 2-1. Programming Model—All Registers

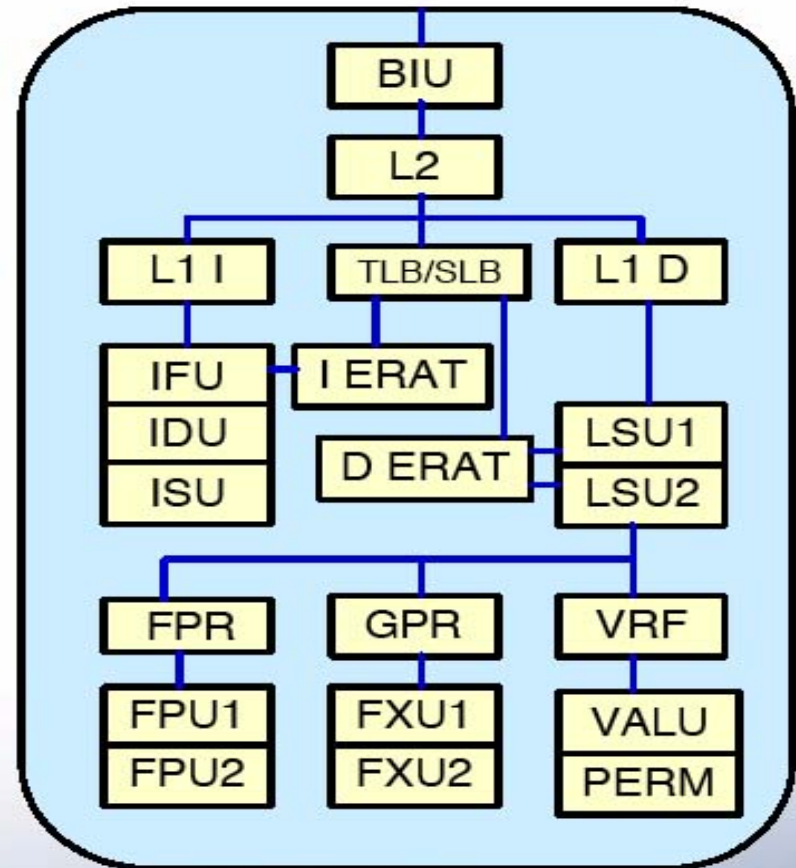


Complexity of CPU cores

PPC 970 Features

PowerPC

- **Instruction pipe**
 - 64KB L1 Inst cache, direct mapped
 - 32 entry I buffer
 - 8 instructions fetch / cycle
- **Branch prediction**
 - Highly accurate dynamic prediction
- **Dispatch, issue**
 - 1 group (4 + branch) / cycle
 - Up to 20 active groups
 - Up to 8 issue / cycle
 - Over 200 instructions in flight
- **Data pipe**
 - 32 KB L1 Data cache, 2-way sa
 - 32 x 64b GPR, FPR
 - 32 x 128b VRF
 - 512KB L2 cache, 8-way sa
 - 8 data prefetch streams



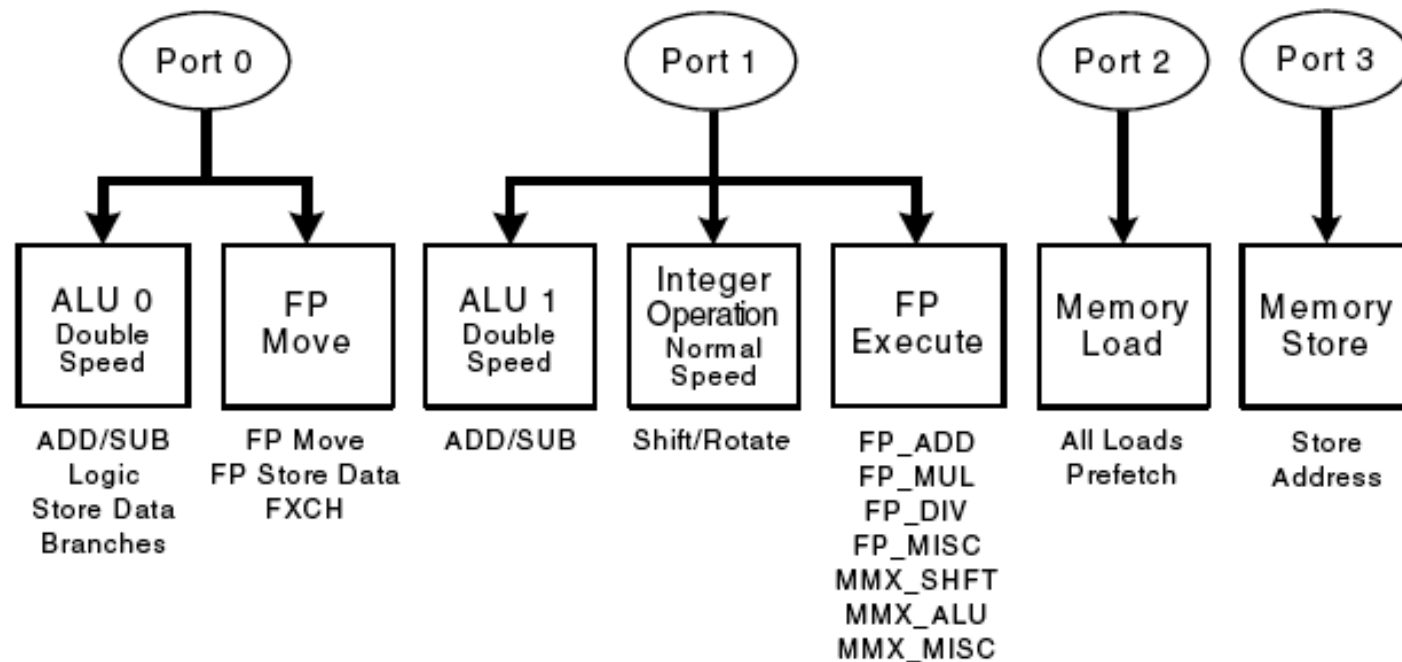
IBM

PowerPC 970 – MPF 2002

Microelectronics

Complexity of execution

Figure 1-4 Execution Units and Ports in the Out-Of-Order Core



Note:

FP_ADD refers to x87 FP, and SIMD FP add and subtract operations
FP_MUL refers to x87 FP, and SIMD FP multiply operations
FP_DIV refers to x87 FP, and SIMD FP divide and square root operations
MMX_ALU refers to SIMD integer arithmetic and logic operations
MMX_SHFT handles Shift, Rotate, Shuffle, Pack and Unpack operations
MMX_MISC handles SIMD reciprocal and some integer operations

OM15151

Initial programming interface

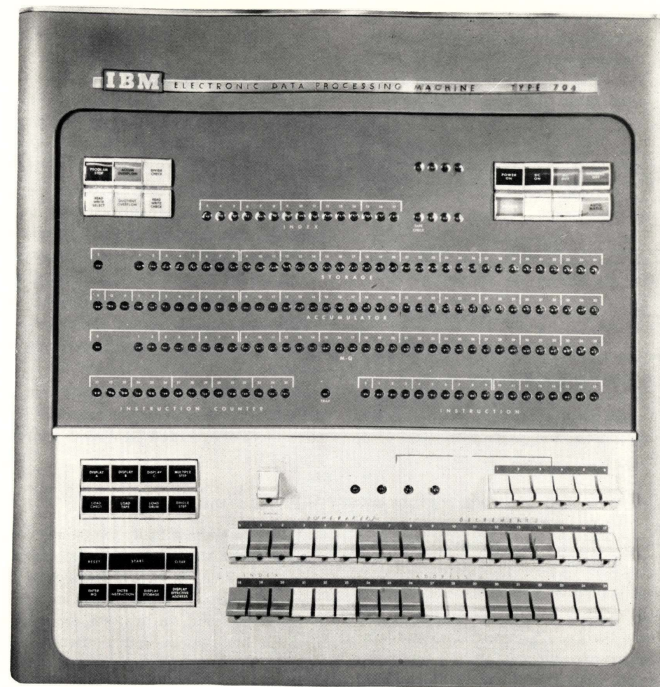


FIGURE 12

points in the program, giving sense instructions (explained under "Instructions") with the addresses of the sense switches causes the calculator to follow one of two courses, depending on which sense switches are depressed. The sense switches are also effective while the calculator is on MANUAL.

Panel Input Switches. There are 36 panel input switches, enabling the operator to insert a word of information into various registers of the calculator while it is on MANUAL. When a panel input switch is down, it represents a 1; when up, it represents a 0.

Index Display Keys. The three index display keys let the operator display the contents of any of the index registers, while the calculator is on MANUAL,

by pressing the key marked with the letter corresponding to the index register in question. For example, to display the contents of index register A, the operator presses the key marked DISPLAY A; the contents of index register A then appears in the index lights. The index registers are automatically displayed until the calculator is returned to automatic operation.

More than one index register can be manually displayed in sequence by pressing the Display A, Display B, and Display C keys, in that order. No return to the automatic mode is necessary.

Load Keys. The load keys let the operator initiate the loading of a self-loading program stored on

Unix

- developed in the early 1970s
- as a simpler version of MULTICS
- today freely available versions of unix-clones for all kinds of machines:
 - Linux
 - FreeBSD, openBSD, NetBSD
 - Darwin
- consider only the common concepts etc.
- remember the man command!

logging in

- each unix machine supports an unlimited number of simultaneous users
- you need an userid and a password to use a unix machine
- if you sit before a unix machine, it'll ask for your userid and password
- if you want to connect to and work on a remote machine, use
- `ssh UID@machine.internet.name`

shells

- unix is command driven
- graphical interfaces (X Windows) are nothing more than programs that run on a unix machine
- they are 1980's add-ons to unix
- the command processors are called 'shells'
- they are just programs themselves
- there are a number of them: ksh, bash, csh, tcsh, sh
- they offer different add-ons to the user, e.g., command line editing and scripting

files

- unix uses a generalized file concept
 - 'normal' files (tex, programs, data etc)
 - devices (disks, tapes, CDROMs, display, mice etc)
 - data & devices on remote computers
- file system is hierarchical (uses sub-directories starting at a root)

files

- file names are nearly arbitrary
- case sensitive (not always, e.g., OSX's HFS+)
- important special characters:
 - `/`: used as directory separator
 - `*`, `?`: wildcard characters used to select groups of files
 - `\`: used to mask special characters
- tip: use alphanumeric characters for files: A-Z a-z
0-9

canonical files/names

- /dev: directory with hardware devices
- * .f: fortran source code files
- * .c: C source code files
- * .tex: T_EX source code files

essential commands

- `man`: gets you help on a command, e.g, `man man`
- `df`: shows all file systems and their used/free space
- `pwd`: shows current directory and it's path starting at the root
- `ls`: lists files in current directory
- `cd`: changes to a different directory

essential commands

- `cat`, `more`, `less`: display content of a (text) file on the terminal
- `rm`: removes a file. Forever. It's gone, bye-bye. No way back.
- `exit`: ends the shell, closes the connection to a unix machine
- `xterm`: starts a new shell in a new X Window (if applicable!)

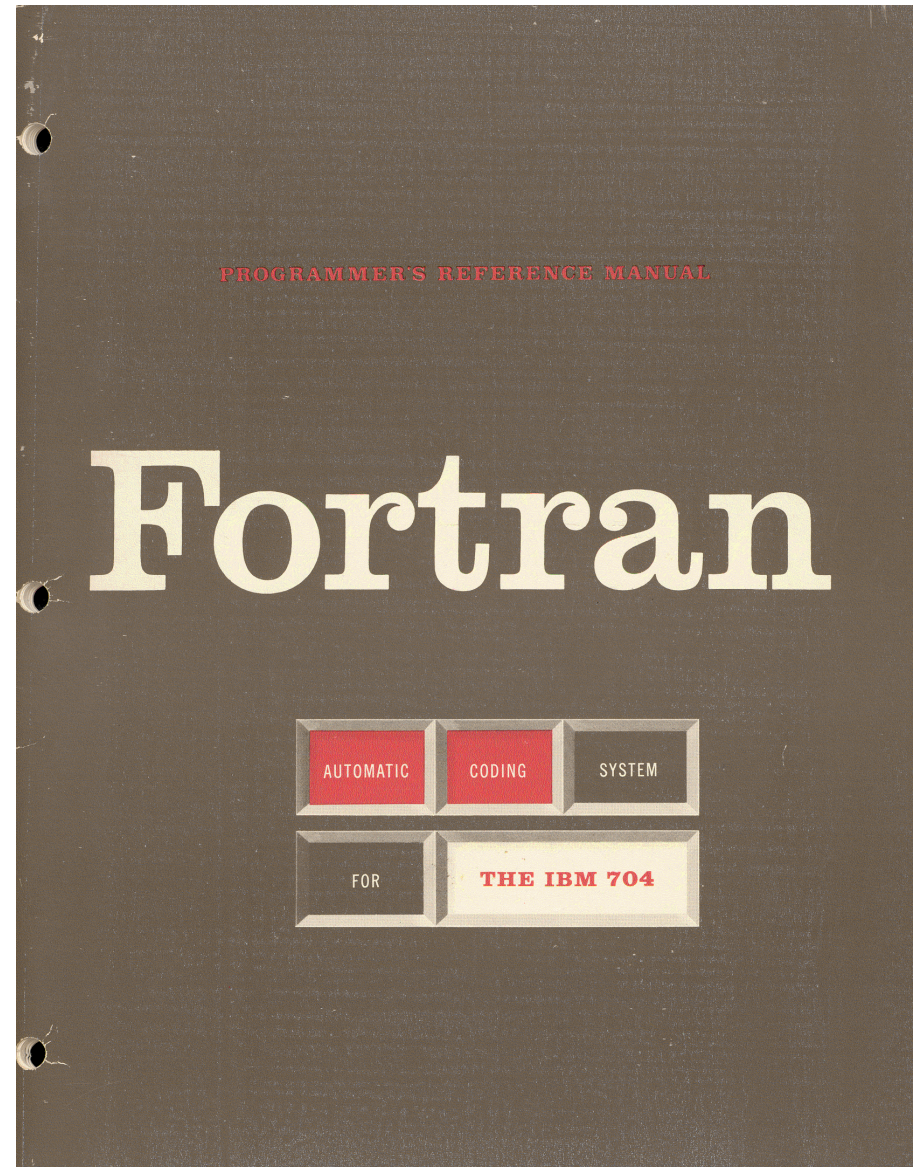
editing files

- there are many different editors available
- `vi` (pronounced as separate letters: “v i”): the standard
- `emacs`: a freely available full-screen text editor.
- `pico`: This is the editor that comes with the `pine` email program

Programming

- direct programming of modern CPUs?
- use high level languages
- historically: FORTRAN (IBM, early 1950's)

Fortran Automatic Coding System

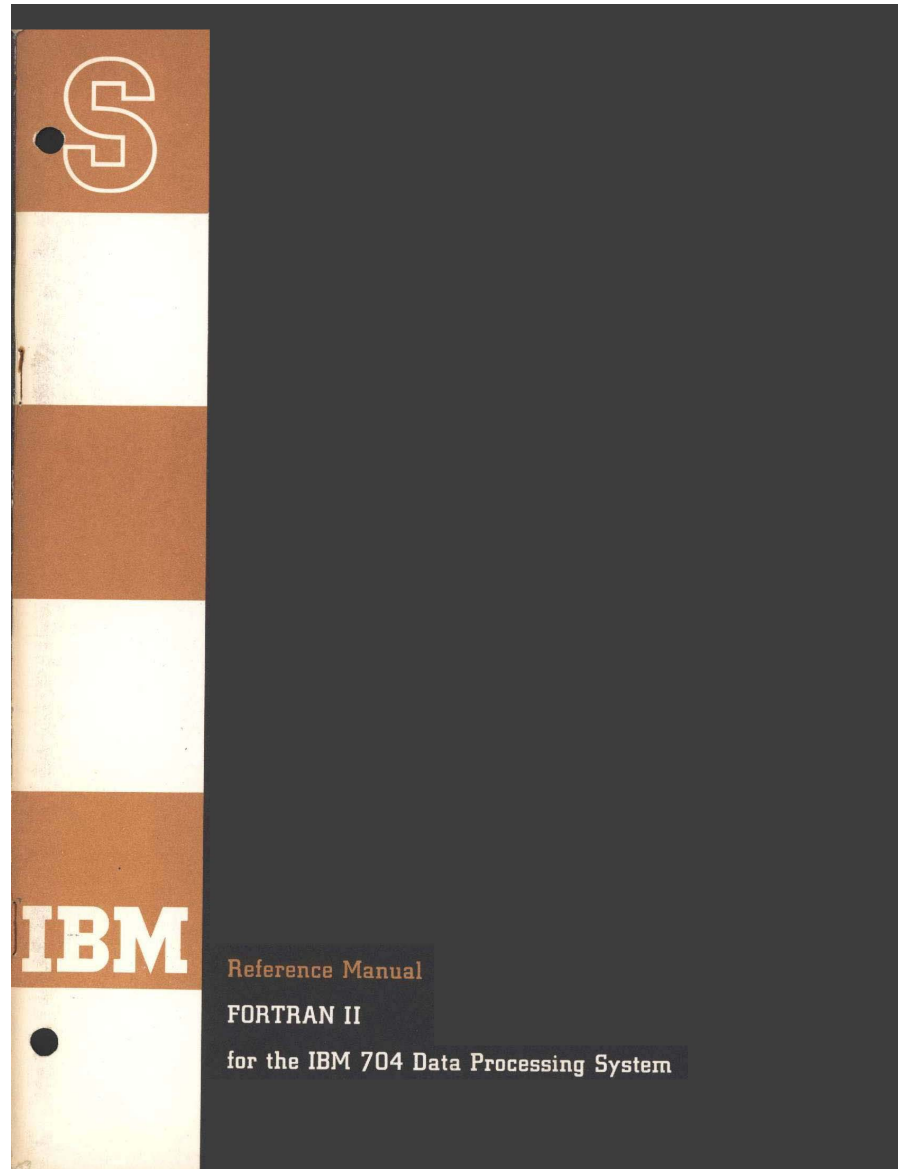


original Fortran Language

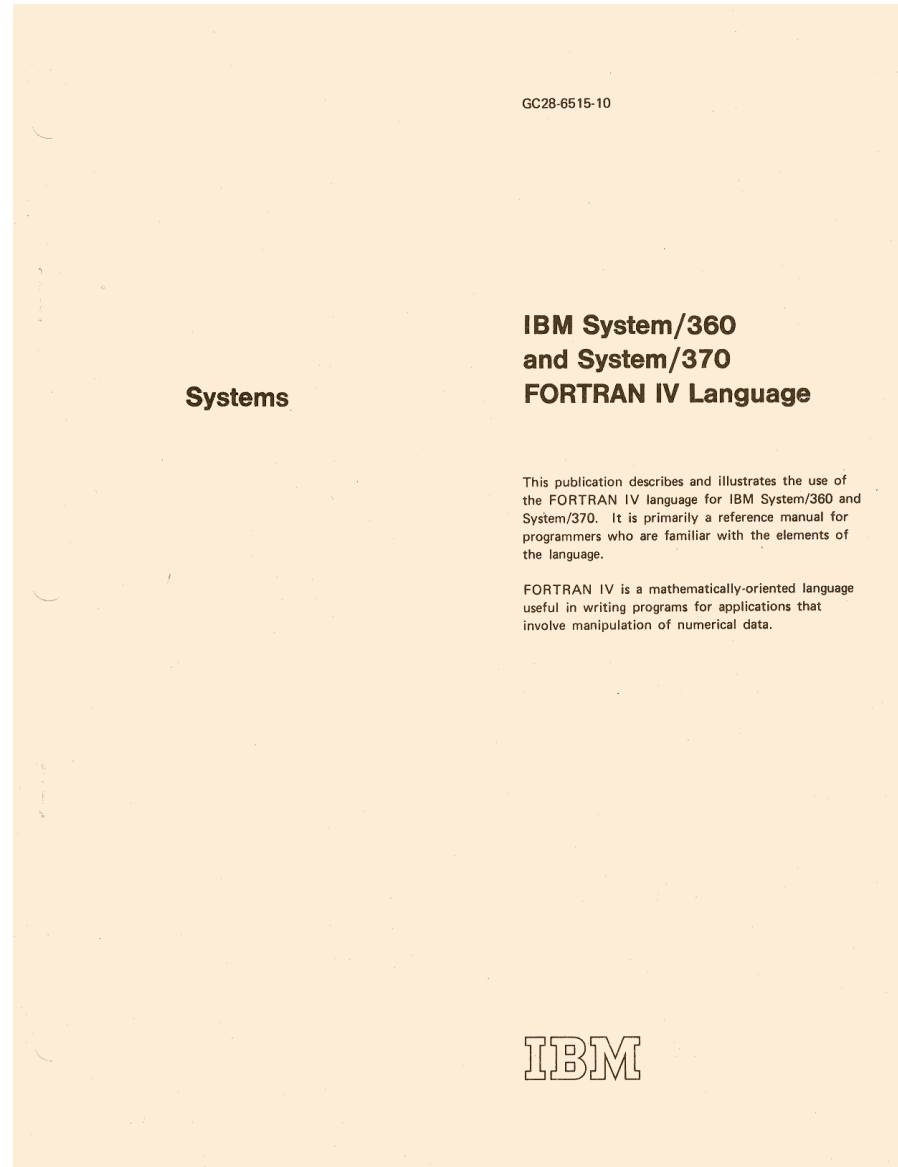
APPENDIX B. TABLE OF FORTRAN STATEMENTS

STATEMENT	NORMAL SEQUENCING
$a = b$	Next executable statement
GO TO n	Statement n
GO TO $n_1, (n_1, n_2, \dots, n_m)$	Statement last assigned
ASSIGN i TO n	Next executable statement
GO TO $(n_1, n_2, \dots, n_m), i$	Statement n_i
IF $(a) n_1, n_2, n_3$	Statement n_1, n_2, n_3 as a less than, =, or greater than 0
SENSE LIGHT i	Next executable statement
IF (SENSE LIGHT i) n_1, n_2	Statement n_1, n_2 as Sense Light i ON or OFF
IF (SENSE SWITCH i) n_1, n_2	" " " " as Sense Switch i DOWN or UP
IF ACCUMULATOR OVERFLOW n_1, n_2	" " " " as Accumulator Overflow trigger ON or OFF
IF QUOTIENT OVERFLOW n_1, n_2	" " " " as MQ Overflow trigger ON or OFF
IF DIVIDE CHECK n_1, n_2	" " " " as Divide Check trigger ON or OFF
PAUSE or PAUSE n	Next executable statement
STOP or STOP n	Terminates program
DO $n i = m_1, m_2$ or DO $n i = m_1, m_2, m_3$	Next executable statement
CONTINUE	" " " "
FORMAT (Specification)	Not executed
READ n , List	Next executable statement
READ INPUT TAPE i , n , List	" " " "
PUNCH n , List	" " " "
PRINT n , List	" " " "
WRITE OUTPUT TAPE i , n , List	" " " "
READ TAPE i , List	" " " "
READ DRUM i, j , List	" " " "
WRITE TAPE i , List	" " " "
WRITE DRUM i, j , List	" " " "
END FILE i	" " " "
REWIND i	" " " "
BACKSPACE i	" " " "
DIMENSION $v, v, v, \dots$	Not executed
EQUIVALENCE $(a, b, c, \dots), (d, e, f, \dots), \dots$	" " " "
FREQUENCY $n(i, j, \dots), m(k, l, \dots), \dots$	" " " "

FORTRAN II (IBM 704)



FORTRAN IV



programming

- 'all' programming languages are available on unix
- writing a program is a multi-stage process:
 - develop and write program source code (text file)
 - use a compiler to translate source to 'object' format
 - link object code with system libraries to produce executable program
- we will use fortran95 as programming language

programming support

- fortran95 compile have different names:
 - gfortran/g95: GNU gcc based Fortran 95, FREE
 - f95: most common, e.g., NAG fortran95
 - ifort: Intel fortran95 compiler for IA32/IA64
 - xlf95: IBM fortran95 compiler for OSX, AIX, Linux on PPCs
- important utility program: make
- important debugger: gdb

Compiler

- the “compiler” is (usually) a set of different programs
 - compiler: translates source code to “intermediate code”
 - optimizer: optimizes intermediate code for performance
 - assembler: translates intermediate code to object code
 - linker: combines object codes with libraries to executable
- libraries: special files with system, mathematical, I/O routines

Compiler

- have a number of options
- these depend on the actual compiler!
- typical:
 - debugging (testing/verifying the code)
 - optimizing (faster execution)
- *you must know the options for your compiler*
- *how to find out?*
 - `man f95`
 - `f96 --help`

HelloWorld.f

```
program hello
```

```
-----
```

```
implicit none
```

```
write(*,*) 'Hello World!'
```

```
stop "die, puny human!"
```

```
end
```

HelloWorld.f

```
program_hello
c-----
implicit_none
c
write(*,*)_Hello_World!'
c
stop_"die,_puny_human!"
c
end
```

make

- make is an “utility”
- aid program development
- uses a `Makefile` to describe how to translate source code to executable
- useful for projects with many sources files that have dependencies

Makefile

```
F90=gfortran
LD=gfortran
FFLAGS= -O5 -fno-automatic -ffixed-form -Wall -fno-second-underscore
DFFLAGS= -g -Wall -fno-automatic -ffixed-form -fbounds-check -fno-second-underscore
LFLAGS=${FFLAGS}

.f.o:
    $(F90) $(FFLAGS) -c $<

all: HelloWorld

clean:
    /bin/rm *.o HelloWorld

HelloWorld: hello.o
    $(LD) -o HelloWorld hello.o $(LFLAGS)
```

make

- attention: 'blanks' before commands are tabs!
- Makefiles can be very complex, depending on the project!
- actual command: make
- in the example: `make clean` removes generated files

problems

1. write/compile/run a 'hello world' program in fortran95
2. write a Makefile to compile/link hello.f