Computer Systems Architecture

Dr Rob Williams

Course text:

"Computer Systems Architecture - a networking approach"
Edition 2
Prentice Hall, 2006
1. CSA - the Hardware / Software Interface

Computer Architecture?

Interaction of h/w & s/w

Layered hierarchy of s/w on a h/w bed
Moore's Law of technological progress

ENTITY decoder8 IS
  PORT (sel: IN std_logic_vector (2 DOWNTO 0); -- select i/p signals
        sig: out std_logic_vector (7 downto 0)); -- eight o/p signals
END decoder8;

ARCHITECTURE rtl OF decoder8 IS
BEGIN
  s <= "0000_0001" WHEN (sel = X"0") ELSE
       "0000_0010" WHEN (sel = X"1") ELSE
       "0000_0100" WHEN (sel = X"2") ELSE
       "0000_1000" WHEN (sel = X"3") ELSE
       "0001_0000" WHEN (sel = X"4") ELSE
       "0010_0000" WHEN (sel = X"5") ELSE
       "0100_0000" WHEN (sel = X"6") ELSE
       "1000_0000";
END rtl;

Modern h/w development: VHDL
Telephone Switch showing the embedded computer

Windows’ file browser
DLL initialization failure
C:\WINNT\System32\KERNEL32.DLL
The process is terminating abnormally

The local ATM gives an error message

Unix set up script or batch file

```bash
% cat .cshrc
umask 077
limit core 0
setenv TERM vt100
setenv PRINTER lw
set prompt = "hostname > "
set history = 25
biff y
mesg n
alias tt99 'setenv DISPLAY TT99:0'
set path = ( . /usr/ucb /usr/bin/X11 /bin /usr/bin /usr/local
set path = ($path /etc /usr/etc /usr/lang /usr/local $home/bin)
```
Original sources of the WWW

The Internet
University LAN

Ethernet packet structure

Williams R, Computer Systems Architecture, Prentice Hall,
Tanenbaum A S, "Structured Computer Organization", Prentice Hall,
Heuring & Jordon, "Computer Systems Design and Architecture", Addison Wesley
Patterson & Hennessy, "Computer Organization & Design: The Hardware/Software Interface", Morgan Kaufmann
Buchanan W, "PC Interfacing, Communications & Windows Programming", Addison Wesley
2. CSA - the von Neumann Inheritance

All under program control

Stored program control

<table>
<thead>
<tr>
<th>Computer</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smart Card</td>
<td>Telephone/credit card</td>
</tr>
<tr>
<td>Microcontroller</td>
<td>Washing machine controller</td>
</tr>
<tr>
<td>Games Console</td>
<td>Interactive entertainment</td>
</tr>
<tr>
<td>Home PC</td>
<td>Web information browsing</td>
</tr>
<tr>
<td>Workstation</td>
<td>Design layouts for circuit boards</td>
</tr>
<tr>
<td>Office Server</td>
<td>Central filing on local network</td>
</tr>
<tr>
<td>Mainframe</td>
<td>Corporate Database</td>
</tr>
<tr>
<td>Supercomputer</td>
<td>Flight simulation studies</td>
</tr>
</tbody>
</table>

Common applications of computers
from HLL: \[ i = j + k; \]

to assembler mnemonics:
- `mov EAX, [12011234]`
- `add EAX, [12011238]`
- `mov [1201123C], EAX`

to machine binary:
- `0010 0000 0011 1001`
- `0001 0010 0000 0001`
- `0001 0010 0011 0100`
- `1101 0000 1011 1001`
- `0001 0010 0000 0001`
- `0001 0010 0011 1000`
- `0010 0011 1100 0000`
- `0001 0010 0000 0001`
- `0001 0010 0011 1100`

---

HLL, assembler & machine code

---

Categories of machine instructions

1. Data Transfer and Manipulation
2. Input / Output
3. Transfer of Program Control
4. Machine Control
Phases of a HLL compiler

Code sharing at different phases
4096 + 2048 + 1024 + 512 + 128 + 32 + 16 + 8 + 1 = 7865

Binary to decimal & decimal to binary conversion

remainders written from right to left

1 1 1 1 1 1 0 1

2 ) 2 3 9 7
1 1 9 8
5 9 9
2 9 9
1 1 4 9
7 4
3 7
1 1 8
9 4
2 1
0

Hex & binary
ASCII code table
```c
#include <stdio.h>

void main() {
    putchar(7);
}
```

### Ring the bell

```c
char letter;
short count;
unsigned int uk_population;
long world_population;

float body_weight;
double building_weight;
long double world_weight;
```

### Data types

<table>
<thead>
<tr>
<th>AIX</th>
<th>OS/2</th>
<th>CDOS</th>
<th>PICK</th>
</tr>
</thead>
<tbody>
<tr>
<td>CICS</td>
<td>PRIMOS</td>
<td>CMS</td>
<td>RSTOS</td>
</tr>
<tr>
<td>CP/M</td>
<td>RSX/11</td>
<td>MSDOS</td>
<td>RTL/11</td>
</tr>
<tr>
<td>George</td>
<td>TDS</td>
<td>IDRIS</td>
<td>THE</td>
</tr>
<tr>
<td>ISIS</td>
<td>UNIX</td>
<td>LYNXOS</td>
<td>Ultrix</td>
</tr>
<tr>
<td>MINIX</td>
<td>VERSADOS</td>
<td>MOP</td>
<td>VM</td>
</tr>
<tr>
<td>MSDOS</td>
<td>VMS</td>
<td>MVS</td>
<td>MS WINDOWS</td>
</tr>
<tr>
<td>Multics</td>
<td>XENIX</td>
<td>OS-9</td>
<td>Linux</td>
</tr>
</tbody>
</table>

### Operating Systems

1. Command line interpreter (CLI), shell script or desktop selections
2. Function calls from within user programs (API)

### Access to O/S facilities
Unix unbuffered, nonblocked keyboard

```c
#include <errno.h>
#include <stdio.h>
#include <sys/termios.h>
#include <unistd.h>
#define TIMEOUT -1

extern int errno;
int sys_nerr;
extern char * sys_errlist[];

void setterm(void) {
    struct termios tty;
    int status;
    status = ioctl(0, TGETS, &tty);
    tty.c_lflag &= ~ICANON;
    tty.c_cc[VMIN] = 1;
    tty.c_cc[VTIME] = 0;
    status = ioctl(0, TSETS, &tty);
    if ( status == -1 ) {
        printf("ioctl error \n");
        perror(sys_errlist[errno]);
        exit();
    }
}
```
Onion layered model for Operating Systems

- **sh** - the original Bourne shell, still popular with administrators for scripts
- **csh** - the C shell, more C-like syntax, and is better for interactive sessions
- **tcsh** - Tenex shell, perhaps the most used interactive shell, emacs keying
- **ksh** - Korn shell, normal issue with Hewlett Packard workstations
- **bash** - bourne-again-shell, a free-ware rework of several shells

Unix command shells
Client-server computing

Xterm

```bash
atarip@pong [50] xterm &
atarip@pong [51] rlogin milly -l rwilliam
Last login: Tue Jul 1 09:22:21 sister
rwilliam@milly >
```
Computers
  Fetch-execute cycle
  Hardware
    CPU
      Arithmetic Logic Unit
      Control Unit
      RISC features
      ARM processor
      Pentium
      Itanium
    Input-output
      Parallel communication
      Serial communication
    Networking
      Local Area Networks
      Ethernet
      USB
      Wide Area Networks
      Other Networks
      Point to point
      Visual output
    Memory
      Memory hierarchy
      Cache and main memory
      Disk filing
    Parallel processing
  Software
    Operating systems
      Unix
      MS Windows
    Tools
      Compilers and assemblers
      Subroutines and stacks
      WIMP
  Users' viewpoints
    Hardware engineer
    HLL programmer
    Systems administrator
    Systems programmer
3. CSA - the Fetch-execute Cycle

Fig 3.2
PC ATX Motherboard, showing the locations of the CPU, memory and I/O card sockets

Motherboard from a PC
Subsystems joined by a bus highway

Point-to-point vs. bus interconnect schemes

CPU has two main component parts
<table>
<thead>
<tr>
<th>ns</th>
<th>μs</th>
<th>ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 000 000 000</td>
<td>1 000 000</td>
<td>1 000</td>
</tr>
</tbody>
</table>

**fetch-execute**
- 10 ns

**logic gate delay**
- 5 ns

**SRAM access**
- 15 ns

**Comparative speeds**

- engine spark: 10 μs, 20 ms
- car engine (3000 rpm): 2-20 μs, 10 ms
- tv line scan: 60 μs, 20 ms
- tv frame: 300 μs, 20 ms
- light: 300 m/μs, 300 ms
- human reaction: 10 ms, 300 ms
The Fetch part of the Fetch-Execute Cycle
The execute part of the Fetch-Execute Cycle

CPU activity for a Sun workstation
Data bus - typically 32 bits wide, but will be increased to 64 bits,
Address bus - 32 bits wide, but will require more very soon,
Control bus - about 15 lines for starting and stopping activities.

System bus has three parts

Timing of synchronous bus activity
Timing of asynchronous bus activity

Timing of multi-phase instructions cycle
Musical interference on FM receivers

Clock speed limitation
Prefetching instructions

Overlapped operations gives greater throughput

Address width determines memory length
16 bit addresses can access $2^{16}$, 65536, 64K locations
20 bit addresses can access $2^{20}$, 1048576, 1M locations
24 bit addresses can access $2^{24}$, 16777216, 16M locations
32 bit addresses can access $2^{32}$, 4294967296, 4G locations
64 bit addresses can access $2^{64}$, 4398046511104, 4E locations

<table>
<thead>
<tr>
<th>dec</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin</td>
<td>0000</td>
<td>0001</td>
<td>0010</td>
<td>0011</td>
<td>0100</td>
<td>0101</td>
<td>0110</td>
<td>0111</td>
<td>1000</td>
<td>1001</td>
<td>1010</td>
<td>1011</td>
<td>1100</td>
<td>1101</td>
</tr>
<tr>
<td>hex</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>
Byte ordering: big endian, little endian

```c
unsigned char  b1[ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 251, 252, 253, 254, 255};
unsigned short b2[ ] = {1, 2, 3, 4, 5, 254, 255, 256, 257, 65532, 65533, 65534, 65535};
unsigned int   b4[ ] = {1, 2, 3, 4, 5, 254, 255, 256, 4095, 4096, 4097, 4294967295};
```
Parallel data input & output ports
4. CSA - the Control Unit (CU)

Inputs
\[
\begin{array}{c|c|c}
A & B & A \text{ AND } B \\
\hline
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

Inputs
\[
\begin{array}{c|c|c}
A & B & A \text{ OR } B \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

Inputs
\[
\begin{array}{c|c|c}
A & B & A \text{ XOR } B \\
\hline
0 & 0 & 0 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

Inputs
\[
\begin{array}{c|c}
A & \text{ NOT } A \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\]

Basic logic gates with truth tables

Using AND for pattern recognition

Detect: 111

Using AND as a data valve
Control
W X Y Z     Data
            A B C D     Out
0 0 0 1     a b c d     a
0 0 1 0     a b c d     b
0 1 0 0     a b c d     c
1 0 0 0     a b c d     d

Data selector, 1 from 4

\[ O = (A \ \text{AND} \ Z) \ \text{OR} \ (B \ \text{AND} \ Y) \ \text{OR} \ (C \ \text{AND} \ X) \ \text{OR} \ (D \ \text{AND} \ W) \]

Data lines
A B C D

Control lines
W X Y Z

Selector line
Y X d c b a

\[ O = (A \ \text{AND} \ (\bar{X} \ \text{AND} \ \bar{Y})) \ \text{OR} \ (B \ \text{AND} \ (X \ \text{AND} \ \bar{Y})) \ \text{OR} \ (C \ \text{AND} \ (\bar{X} \ \text{AND} \ Y)) \ \text{OR} \ (D \ \text{AND} \ (X \ \text{AND} \ Y)) \]

2-to-4 line decoder, 1-out-of-4 line selector
\[ O_1 = (\bar{i}_3 \text{ AND } i_2 \text{ AND } i_1) \text{ OR } (\bar{i}_4 \text{ AND } i_3 \text{ AND } \bar{i}_2) \text{ OR } (i_4 \text{ AND } \bar{i}_3 \text{ AND } i_2 \text{ AND } \bar{i}_1) \]

Data multiplexor, 1 from 4
$O_1 = (\overline{i_3} \cdot i_2 \cdot i_1) + (\overline{i_4} \cdot i_3 \cdot \overline{i_2}) + (i_4 \cdot \overline{i_3} \cdot i_2 \cdot \overline{i_1})$

**Sum of Products solution**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>NAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**The 2 input NAND gate**
Programmable Logic Array (PLA)
Level Crossing

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Lights</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Y</td>
<td>R A G</td>
</tr>
<tr>
<td>0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>1 1</td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

Cross Roads

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Lights</th>
</tr>
</thead>
<tbody>
<tr>
<td>X Y Z</td>
<td>R A G</td>
</tr>
<tr>
<td>W E N</td>
<td>W E S</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 0 1</td>
</tr>
</tbody>
</table>

R = \bar{X} 
A = Y 
G = X \text{ AND } \bar{Y} 

Traffic light controllers

\begin{align*}
R &= (X \text{ AND } Y) \\
A &= (\bar{X} \text{ AND } Z) \\
G &= (\bar{X} \text{ AND } Y \text{ AND } \bar{Z})
\end{align*}
### Inputs vs. Outputs Table

<table>
<thead>
<tr>
<th>Enable</th>
<th>Select</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G_1$</td>
<td>$G_2$</td>
<td>$CBA$</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>01234567</td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>01234567</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>01234567</td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>01234567</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>01234567</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>01234567</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>01234567</td>
</tr>
</tbody>
</table>

---

**3 to 8 line decoder**

---

CSA Rob Williams

Pearson Education (c) 2006

CSA ch 04 - p 40
<table>
<thead>
<tr>
<th>Inputs</th>
<th>LEDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>W X Y Z</td>
<td>a b c d e f g</td>
</tr>
<tr>
<td>0 0 0 0</td>
<td>1 1 1 1 1 1 0</td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 1 1 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>0 0 1 1</td>
<td>1 1 1 1 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 1 1 0 0 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 0 1 1 0 1 1</td>
</tr>
<tr>
<td>0 1 1 0</td>
<td>0 0 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 1 1 0 0 0 0</td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>1 1 1 0 1 1 1</td>
</tr>
</tbody>
</table>

\[ a = (\bar{W} \text{ AND } \bar{X} \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (\bar{W} \text{ AND } X \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (W \text{ AND } X \text{ AND } Y \text{ AND } Z) \]

\[ b = (W \text{ AND } X \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (\bar{W} \text{ AND } X \text{ AND } Y \text{ AND } Z) \]

\[ c = W \text{ AND } \bar{X} \text{ AND } Y \text{ AND } Z \]

\[ d = (W \text{ AND } \bar{X} \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (\bar{W} \text{ AND } X \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (W \text{ AND } X \text{ AND } Y \text{ AND } Z) \]

\[ e = (\bar{W} \text{ AND } \bar{X} \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (W \text{ AND } X \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (W \text{ AND } X \text{ AND } Y \text{ AND } Z) \]

\[ f = (\bar{W} \text{ AND } \bar{X} \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (\bar{W} \text{ AND } X \text{ AND } Y \text{ AND } Z) \text{ OR } (\bar{W} \text{ AND } X \text{ AND } Y \text{ AND } Z) \text{ OR } (W \text{ AND } X \text{ AND } Y \text{ AND } Z) \]

\[ g = (\bar{W} \text{ AND } \bar{X} \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (\bar{W} \text{ AND } X \text{ AND } \bar{Y} \text{ AND } Z) \text{ OR } (\bar{W} \text{ AND } X \text{ AND } Y \text{ AND } Z) \text{ OR } (\bar{W} \text{ AND } X \text{ AND } Y \text{ AND } Z) \]

---

**Binary to 7-segment decoder**

CSA Rob Williams

Pearson Education (c) 2006
Washing machine Finite State Diagram (FSD)

1 rev per \( \frac{1}{2} \) hr → 2 rph → \( \frac{2}{3600} \) rps → \( \frac{1}{1800} \) Hz → \( \frac{1000}{1800} \) mHz → 0.55 mHz
Washing machine sequence controller (FSM)
Washing machine controller with conditional branching
Hardware logic Control Unit (RISC)
Microcoded Control Unit (CISC)
5. CSA - the Arithmetic & Logic Unit (ALU)

\[(X \text{ AND } Y) \equiv (\bar{X} \text{ OR } \bar{Y})\]

\[(X \text{ OR } Y) \equiv (\bar{X} \text{ AND } \bar{Y})\]

while ( ! dog && ! cat )

    { plant_flowers( ); }

while ( ! (dog || cat ) )

    { plant_flowers( ); }

De Morgan’s equivalences

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>ADD</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

Alternative Half Adder (2 inputs) circuits
More Half Adder circuits

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>C</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Full Adder (3 inputs) circuit
4 bit parallel adder circuit
Negative integers using Twos Compliment format

To form a two’s compliment negative:
Take the positive number,
invert all the bits,
add 1.

ALU with positive and negative capability

CSA Rob Williams
Pearson Education (c) 2006
Barrel Shifter circuit for Shifts & Rotates
Integer multiplication by Shift and Add

/* function to multiply two 16 bit positive integers returning a 32 bit result, using only integer addition and shift operators */

int multiply(int a, int c)
{
    int i;

    c = c << 16;

    for (i=0; i<16; i++)
    {
        if (a & 1) { a += c ];
        a = a >>1;
    }
    return a;
8 x 8 multiply using two 16 bit registers

CSA Rob Williams
Pearson Education (c) 2006
ALU with data registers
Example integer ALU component

CSA Rob Williams

Pearson Education (c) 2006
float net_cost, tot_cost, price;
float vat = 0.175;
int items;
    net_cost = price * items;
    tot_cost = net_cost + net_cost * vat;

Floats & integers in HLL programming
Floating-point numbers, in IEEE 754 32 bit format, appear in memory as:

<table>
<thead>
<tr>
<th>S</th>
<th>Exponent</th>
<th>Mantissa</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10001001</td>
<td>001101001010010000000000</td>
</tr>
<tr>
<td>1</td>
<td>10000000</td>
<td>101010000000000000000000</td>
</tr>
<tr>
<td>0</td>
<td>01111011</td>
<td>100000000000000000000000</td>
</tr>
</tbody>
</table>

To manually converting a decimal float into a IEEE binary float:

1. Convert the integer part into binary.
2. Convert the fractional part into binary, noting the 1/2, 1/4, 1/8, 1/16 pattern!

... 128 64 32 16 8 4 2 1 • 0.5 0.25 0.125 0.0625 0.03125 ...

3. Normalize by moving the binary point to produce the format: 1.something with a positive or negative shift number.
4. Delete the leading 1, and extend the left bits with 0s to give a 23 bit mantissa.
5. Add 127 to the shift number to give the 8 bit exponent.
/ * floatit.c - to write a real number into a file for viewing */

#include <stdio.h>

int main( )
{
    FILE *fp;
    float f = 231.125;
    if (fp = fopen ("float_data", "w"))
    {
        fwrite(&f, 4, 1, fp);
    }
    return 0;
}
The hex value 43 67 20 00 is the 32 bit floating-point number:

```
0 100 0011 0110 0111 0010 0000 0000 0000
```

- **sign**: 1 bit
- **exponent**: 8 bits
- **mantissa**: 23 bits of the 24bit offset format

The range and precision of the various floating-point formats are as follows:

<table>
<thead>
<tr>
<th>Format</th>
<th>Sign bit</th>
<th>Exponent bit</th>
<th>Mantissa bit</th>
<th>Range</th>
<th>Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 bit</td>
<td>8 bit</td>
<td>24 bit</td>
<td></td>
<td>(1 in 16 x 10⁶)</td>
<td></td>
</tr>
<tr>
<td>64 bit</td>
<td>11 bit</td>
<td>53 bit</td>
<td></td>
<td>(1 in 8 x 10¹⁵)</td>
<td></td>
</tr>
<tr>
<td>128 bit</td>
<td>15 bit</td>
<td>64 bit</td>
<td></td>
<td>(1 in 16 x 10¹⁸)</td>
<td></td>
</tr>
</tbody>
</table>
6. CSA - the Memory

Write-once, Read-many memory cell

<table>
<thead>
<tr>
<th>$Q_t$</th>
<th>$S_t$</th>
<th>$R_t$</th>
<th>$\bar{Q}_{t+1}$</th>
<th>$\bar{Q}_{t+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>illegal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>illegal</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

S-R Latch, 1 bit static memory
Cat IN-OUT indicator using an S-R latch

Different types of memory

Mitsubishi, M5L27512K-2
64 kbytes EPROM
200 nsec access time

Micron, MT46V128M8TG-6T
128 Mbyte Dynamic RAM
167 MHz operation, 6 ns access
Flip-flops used for frequency division
Dynamic ram (DRAM) single cell and memory array
16 MByte, 50ns access, 32 bit, 72 pin SIMM card

64 MByte, 100 MHz clock, 64 bit, 168 pin DIMM card

72 pin SIMM and 168 pin DIMM, DRAM Modules

Remember: $2^{20} = 1\text{M}$, so: $2^{22} = 4\text{M}$
<table>
<thead>
<tr>
<th>Pin</th>
<th>1</th>
<th>Vss</th>
<th>43</th>
<th>Vss</th>
<th>85</th>
<th>Vss</th>
<th>127</th>
<th>Vss</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>DQO</td>
<td>44</td>
<td>NC</td>
<td>50</td>
<td>NC</td>
<td>51</td>
<td>NC</td>
<td>52</td>
</tr>
<tr>
<td>3</td>
<td>DQ1</td>
<td>45</td>
<td>CS2</td>
<td>46</td>
<td>DQM2</td>
<td>47</td>
<td>DQM3</td>
<td>48</td>
</tr>
<tr>
<td>4</td>
<td>DQ2</td>
<td>49</td>
<td>Vcc</td>
<td>52</td>
<td>Vcc</td>
<td>53</td>
<td>Vcc</td>
<td>54</td>
</tr>
<tr>
<td>5</td>
<td>DQ3</td>
<td>50</td>
<td>NC</td>
<td>53</td>
<td>NC</td>
<td>54</td>
<td>NC</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>Vcc</td>
<td>51</td>
<td>Vcc</td>
<td>54</td>
<td>Vcc</td>
<td>55</td>
<td>Vcc</td>
<td>56</td>
</tr>
<tr>
<td>7</td>
<td>DQ4</td>
<td>52</td>
<td>NC</td>
<td>56</td>
<td>NC</td>
<td>57</td>
<td>NC</td>
<td>58</td>
</tr>
<tr>
<td>8</td>
<td>DQ5</td>
<td>53</td>
<td>NC</td>
<td>57</td>
<td>NC</td>
<td>58</td>
<td>NC</td>
<td>59</td>
</tr>
<tr>
<td>9</td>
<td>DQ6</td>
<td>54</td>
<td>NC</td>
<td>59</td>
<td>NC</td>
<td>60</td>
<td>NC</td>
<td>61</td>
</tr>
<tr>
<td>10</td>
<td>DQ7</td>
<td>55</td>
<td>NC</td>
<td>60</td>
<td>NC</td>
<td>61</td>
<td>NC</td>
<td>62</td>
</tr>
<tr>
<td>11</td>
<td>DQ8</td>
<td>56</td>
<td>NC</td>
<td>62</td>
<td>NC</td>
<td>63</td>
<td>NC</td>
<td>64</td>
</tr>
<tr>
<td>12</td>
<td>DQ9</td>
<td>57</td>
<td>NC</td>
<td>63</td>
<td>NC</td>
<td>64</td>
<td>NC</td>
<td>65</td>
</tr>
<tr>
<td>13</td>
<td>DQ10</td>
<td>58</td>
<td>NC</td>
<td>65</td>
<td>NC</td>
<td>66</td>
<td>NC</td>
<td>67</td>
</tr>
<tr>
<td>14</td>
<td>DQ11</td>
<td>59</td>
<td>Vcc</td>
<td>67</td>
<td>Vcc</td>
<td>68</td>
<td>Vcc</td>
<td>69</td>
</tr>
<tr>
<td>15</td>
<td>DQ12</td>
<td>60</td>
<td>NC</td>
<td>69</td>
<td>Vcc</td>
<td>70</td>
<td>Vcc</td>
<td>71</td>
</tr>
<tr>
<td>16</td>
<td>DQ13</td>
<td>61</td>
<td>NC</td>
<td>70</td>
<td>Vcc</td>
<td>72</td>
<td>Vcc</td>
<td>73</td>
</tr>
<tr>
<td>17</td>
<td>DQ14</td>
<td>62</td>
<td>NC</td>
<td>72</td>
<td>Vcc</td>
<td>73</td>
<td>Vcc</td>
<td>74</td>
</tr>
<tr>
<td>18</td>
<td>DQ15</td>
<td>63</td>
<td>NC</td>
<td>74</td>
<td>Vcc</td>
<td>74</td>
<td>Vcc</td>
<td>75</td>
</tr>
<tr>
<td>19</td>
<td>DQ16</td>
<td>64</td>
<td>Vcc</td>
<td>75</td>
<td>Vcc</td>
<td>75</td>
<td>Vcc</td>
<td>76</td>
</tr>
<tr>
<td>20</td>
<td>DQ17</td>
<td>65</td>
<td>NC</td>
<td>76</td>
<td>Vcc</td>
<td>76</td>
<td>Vcc</td>
<td>77</td>
</tr>
<tr>
<td>21</td>
<td>DQ18</td>
<td>66</td>
<td>NC</td>
<td>77</td>
<td>Vcc</td>
<td>77</td>
<td>Vcc</td>
<td>78</td>
</tr>
<tr>
<td>22</td>
<td>DQ19</td>
<td>67</td>
<td>NC</td>
<td>78</td>
<td>Vcc</td>
<td>78</td>
<td>Vcc</td>
<td>79</td>
</tr>
<tr>
<td>23</td>
<td>DQ20</td>
<td>68</td>
<td>Vcc</td>
<td>79</td>
<td>Vcc</td>
<td>79</td>
<td>Vcc</td>
<td>80</td>
</tr>
<tr>
<td>24</td>
<td>DQ21</td>
<td>69</td>
<td>NC</td>
<td>80</td>
<td>NC</td>
<td>81</td>
<td>NC</td>
<td>82</td>
</tr>
<tr>
<td>25</td>
<td>DQ22</td>
<td>70</td>
<td>NC</td>
<td>82</td>
<td>NC</td>
<td>83</td>
<td>DC</td>
<td>84</td>
</tr>
<tr>
<td>26</td>
<td>DQ23</td>
<td>71</td>
<td>NC</td>
<td>84</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>27</td>
<td>WE</td>
<td>72</td>
<td>DQS</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>28</td>
<td>DQM0</td>
<td>73</td>
<td>Vcc</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>29</td>
<td>DQM1</td>
<td>74</td>
<td>Vcc</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>30</td>
<td>CSO</td>
<td>75</td>
<td>VCC</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>31</td>
<td>NC</td>
<td>76</td>
<td>Vcc</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>32</td>
<td>Vss</td>
<td>77</td>
<td>DQ28</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>33</td>
<td>AO</td>
<td>78</td>
<td>DQ29</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>34</td>
<td>A2</td>
<td>79</td>
<td>DQ30</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>35</td>
<td>A4</td>
<td>80</td>
<td>DQ31</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>36</td>
<td>Vcc</td>
<td>81</td>
<td>DQ32</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>37</td>
<td>A8</td>
<td>82</td>
<td>DQ33</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>38</td>
<td>A10/AP</td>
<td>83</td>
<td>Vss</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>39</td>
<td>BA1</td>
<td>84</td>
<td>Vcc</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>40</td>
<td>Vcc</td>
<td>85</td>
<td>DQ34</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>41</td>
<td>Vcc</td>
<td>86</td>
<td>DQ35</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
<tr>
<td>42</td>
<td>CLKO</td>
<td>87</td>
<td>DQ36</td>
<td>85</td>
<td>DC</td>
<td>84</td>
<td>DC</td>
<td>85</td>
</tr>
</tbody>
</table>

## Pin assignments for a 168 pin SDRAM DIMM

CSA Rob Williams
Pearson Education (c) 2006
Ideal memory configuration

<table>
<thead>
<tr>
<th>Device</th>
<th>Size</th>
<th>Pins</th>
<th>32 bit address bus</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROM1</td>
<td>1MB</td>
<td>20</td>
<td>0000 0000 xxx</td>
<td>0000 0000 - 000F FFFF</td>
</tr>
<tr>
<td>RAM1</td>
<td>16MB</td>
<td>24</td>
<td>0000 0001 ++++</td>
<td>0100 0000 - 01FF FFFF</td>
</tr>
<tr>
<td>RAM2</td>
<td>16MB</td>
<td>24</td>
<td>0000 0010 ++++</td>
<td>0200 0000 - 02FF FFFF</td>
</tr>
<tr>
<td>RAM3</td>
<td>16MB</td>
<td>24</td>
<td>0000 0011 ++++</td>
<td>0300 0000 - 03FF FFFF</td>
</tr>
<tr>
<td>RAM4</td>
<td>16MB</td>
<td>24</td>
<td>0000 0100 ++++</td>
<td>0400 0000 - 04FF FFFF</td>
</tr>
</tbody>
</table>

+ address line used directly for internal selection
x line ignored, indicates partial (degenerate) addressing
0 must be 0 for chip selection
1 must be 1 for chip selection

Memory map for a small computer system

Memory Schematic showing the Decoding Circuit

CSA Rob Williams
Pearson Education (c) 2006
4 Gbyte Memory Organisation
Memory layout for a Memory-mapped I/O Scheme

ori.b #bmask,OP_reg ; logical OR a mask to set a port bit
andi.b #$f7,OP_reg ; logical AND a mask to clear a port bit
asl.b (a5) ; shift port bits left for display purposes
not.b OP_reg ; shift port bits right for display purposes
bclr #1,OP_reg ; test a port bit and leave it 0
bset #2, (a2) ; test a port bit and leave it 1
Memory and I/O layout for an I/O-mapped scheme
7. CSA - the Intel Pentium

Socket 478 for the Pentium 4

Slot A processor card with a Pentium II
Pentium subsystems schematic
<table>
<thead>
<tr>
<th>Name</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>P24T</td>
<td>486 Pentium OverDrive, 63 or 83MHz, Socket 3</td>
</tr>
<tr>
<td>P54C</td>
<td>Classic Pentium 75-200MHz, Socket 517, 3.3v</td>
</tr>
<tr>
<td>P55C</td>
<td>Pentium NWX 166-266MHz, Socket 7, 2.8v</td>
</tr>
<tr>
<td>P54CTB</td>
<td>Pentium MMX OverDrive 125+, Socket 517, 3.3v</td>
</tr>
<tr>
<td>Tillomook</td>
<td>Mobile Pentium MMX 0.25 $\mu$m, 166-266MHz, 1.8v</td>
</tr>
<tr>
<td>P6</td>
<td>Pentium Pro, Socket 8</td>
</tr>
<tr>
<td>Klamath</td>
<td>Original Pentium II, 0.35 $\mu$m, Slot-I</td>
</tr>
<tr>
<td>Deschutes</td>
<td>Pentium 11, 0.25 $\mu$m, Slot 1, 256 Kbyte LII cache</td>
</tr>
<tr>
<td>Covington</td>
<td>Celeron PII, Slot-I, with no L2 cache</td>
</tr>
<tr>
<td>Mendocino</td>
<td>Celeron, PII with 28 Kbyte L2 cache on die</td>
</tr>
<tr>
<td>Dixon</td>
<td>Mobile Pentium IIPE, 256 Kbyte on-die L2 cache</td>
</tr>
<tr>
<td>Katmai</td>
<td>Pentium III, PII with SSE instructions</td>
</tr>
<tr>
<td>Willamette</td>
<td>Pentium III, on-die L2</td>
</tr>
<tr>
<td>Tanner</td>
<td>Pentium 111 Xeon</td>
</tr>
<tr>
<td>Cascades</td>
<td>PIII, 0.18 $\mu$m, on-die L2</td>
</tr>
<tr>
<td>Merced</td>
<td>P7, First IA-64 processor, on-die 12, 0.18 $\mu$m</td>
</tr>
<tr>
<td>McKinley</td>
<td>1 GHz, Improved Merced, IA-64, 0.18 $\mu$m, copper interconnects</td>
</tr>
<tr>
<td>Foster</td>
<td>Improved PIII, IA-32</td>
</tr>
<tr>
<td>Madison</td>
<td>Improved McKinley, IA-64, 0.13 $\mu$m</td>
</tr>
</tbody>
</table>
i80x86/Pentium CPU Register Set
MOV AX,1234H ; load constant value 4660 into 32 bit accumulator
INC EAX ; add 1 to accumulator value
CMP AL,'Q' ; compare the ASCII Q with the LS byte value in EAX
MOV maxval,EAX ; store accumulator value to memory variable "maxval"
DIV DX ; divide accumulator by value in 16 bit D register

**EBX:** Base registers hold addresses pointing to data structures, such as arrays in memory.

- LEA EBX,marks ; initialize EBX with address of the variable "marks"
- MOV AL,[EBX] ; get byte value into AL using EBX as a memory pointer
- ADD EAX,EBX ; add 32 bits from EBX into accumulator
- MOV EAX,table[BX] ; take 32 bit value from the "table" array using the value in BX as the array index

**ECX:** The Count register has a special role as a counter in loops or bit shifting operations.

- MOV ECX,100 ; initialize ECX as the FOR loop index
- ... ; symbolic address label
- LOOP for1 ; decrement ECX, test for zero, JMP back if non-zero

**EDX:** The Data register can be involved during input/output data transfers or when executing integer multiplication and division. Otherwise it is generally available for holding variables.

- IN AL,DX ; input byte value from port, with 16 bit port address in DX
- MUL DX ; multiply A by value in D

**ESI:** Source Index register is a pointer for string or array operations within the Data Segment.

- LEA ESI,dtable ; initialize SI with memory address of variable "dtable"
- MOV AX,[EBX+ESI] ; get word using Base address and Index register

**EDI:** Destination Index register is a pointer for string or array operations within the Data Segment.

- MOV [EDI],[ESI] ; moves a 32 bit word from source to destination locations in memory
**EBP:** The Stack Base Pointer register is used as the stack frame pointer to support HLL procedure operations. It is taken as an offset within the Stack Segment.

```
ENTER 16 ; saves EBP on stack, copies ESP into EBP, and subtracts 16 from ESP
```

**EIP:** The Instruction Pointer (Program Counter) holds the offset address of the next instruction within the current Code Segment.

```
JMP errors ; forces a new address into EIP
```

**ESP:** The Stack Pointer holds the offset address of the next item available on the stack within the current Stack Segment.

```
CALL subdo ; call a subroutine (subdo), storing return address on stack
PUSH EAX ; save 32bit value in accumulator on stack
```

**EFLAG:** Flag Register contains CPU status flags, implicated in all conditional instructions.

```
JGE back1 ; tests sign flag for conditional jump
LOOP backagain ; tests zero flag for loop exit condition
```

**CS - GS:** These 16 bit Segment Selector registers were originally introduced to expand the addressing range of the i8086 processor while maintaining a 16 bit IP. The Segment Register is added to the EIP register to form a 32 bit address.

**CSDCR - GSDCR:** 64 bit Code Segment Descriptor Cache Register holds the current Code Segment Descriptor, which includes: Base address, size Limit and Access permissions. The Segment Descriptor is obtained from either the Global or Local Descriptor Tables.

**TR:** The Task Register holds the 16 bit segment selector, the 32 bit base address, the 16 bit size limit and the descriptor attributes for the current task. It references a TSS descriptor in the Global Descriptor Table (GDT). When a task switch occurs, the Task Register is automatically reloaded.

**IDTR:** The 48bit Interrupt Descriptor Table Register holds the base address and size limit of the current Interrupt Vector Table (IVT).

**GDTR:** The Global Descriptor Table Register holds the segment descriptors which point to universally available segments and to the tables holding the Local Descriptors.

**LDTR:** Each task can use a Local Descriptor Table in addition to the Global

---

CSA Rob Williams  
Pearson Education (c) 2006  
CSA ch 07 - p 78
Descriptor Table. This register indicates which entry in the Local Segment Descriptor Table to use.

**CR3:** This Control Register points to the directory table for the Paging Unit.

**CR2:** This Control Register points to the routine which handles page faults which occur when the CPU attempts to access an item at an address which is located on a non-resident memory page. The service routine will instigate the disk operation to bring the page back into main memory from disk.
i8080 CPU Register Set from 1975

1. data movement (copying)
2. data input/output operations
3. data manipulation
4. transfer of control
5. machine supervision

Classes of CPU instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MOV</td>
<td>copies data from location to location, register or memory</td>
</tr>
<tr>
<td>LEA</td>
<td>load effective address</td>
</tr>
<tr>
<td>CALL</td>
<td>calls to a subroutine</td>
</tr>
<tr>
<td>RET</td>
<td>return from a subroutine</td>
</tr>
<tr>
<td>PUSH</td>
<td>push an item onto the stack, possibly as a subroutine parameter</td>
</tr>
<tr>
<td>POP</td>
<td>pop an item off the stack</td>
</tr>
<tr>
<td>INC/DEC</td>
<td>increment or decrement</td>
</tr>
<tr>
<td>ADD</td>
<td>arithmetic integer addition</td>
</tr>
<tr>
<td>SUB</td>
<td>arithmetic subtraction for 2s complement integers</td>
</tr>
<tr>
<td>CMP</td>
<td>compare 2 values, a subtract with no result, only setting flags</td>
</tr>
<tr>
<td>AND/OR/XOR</td>
<td>logical operators</td>
</tr>
<tr>
<td>TEST</td>
<td>bit testing</td>
</tr>
<tr>
<td>JZ</td>
<td>conditional jump</td>
</tr>
<tr>
<td>LOOP</td>
<td>implements a FOR loop by decrementing the CX register</td>
</tr>
<tr>
<td>ENTER</td>
<td>sets up a subroutine (procedure) stack frame</td>
</tr>
<tr>
<td>LEAVE</td>
<td>cleans up a stack frame on exit from a subroutine</td>
</tr>
<tr>
<td>JMP</td>
<td>a dreaded jump instruction</td>
</tr>
<tr>
<td>INT</td>
<td>software interrupt to get into an operating system routine</td>
</tr>
</tbody>
</table>
1. the action or operation of the instruction,
2. the "victims" or operands involved,
3. where the result is to go.

Pentium instruction code fields

CSA Rob Williams
Pearson Education (c) 2006
03 C3        ADD AX,BX

0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1

<table>
<thead>
<tr>
<th>ADD  op</th>
<th>Word</th>
<th>AX</th>
<th>BX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dest</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reg-Reg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mode</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination source</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

66 B8 00 00 00 00 00 12 00        MOV EAX,12H

0 1 1 0 0 1 1 0 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

<table>
<thead>
<tr>
<th>32bit prefix</th>
<th>MOV  op</th>
<th>Word</th>
<th>AX</th>
<th>immediate data</th>
</tr>
</thead>
</table>

3C 71        CMP AL,'q'

0 0 1 1 1 1 0 0 0 1 1 1 0 0 0 1

<table>
<thead>
<tr>
<th>CMP  A op</th>
<th>Byte</th>
<th>Immediate data</th>
</tr>
</thead>
</table>

ID: identification flag for CPUID availability
VIP: virtual interrupt pending
VI: virtual interrupt active
AC: alignment check
VM: virtual 8086 mode active
RFR: resume task after breakpoint interrupt
NT: nested task
IOPL: i/o privilege level
O: arithmetic overflow error
D: direction of accessing string arrays
IE: external interrupt enable
T: trap, single step debugging, generates an INT #1 after each instruction
S: sign, MS bit value
Z: zero, result being zero
A: auxiliary carry, used by BCD arithmetic on 4 LS bits
P: parity, operand status
C: carry, indicates an arithmetic carry or borrow result

CPU status flag register

CSA ch 07 - p 82

CSA Rob Williams
Pearson Education (c) 2006
CMP sets the Z flag

CPU EFLAG Register

Z

JZ tests the Z flag

Data Register Direct

MOV EAX, EBX

Immediate Operand (IP indirect)

MOV EAX, 1234

Memory Direct

MOV EAX, [var1]

Address Register Direct

LEA EBX, var1

Register Indirect

MOV EAX, [EBX]

Indexed Register Indirect with displacement

MOV EAX, [table+EBP+ESI]

MOV EAX, table[ESI]
Parallelization by pipelined operation
From Main Memory

8kB Data Cache

LI

8kB Code Cache

F/Point

V-pipe

U-pipe decoder
Editor Window with source code breakpoint mark and IP index mark

Editor Window with source code breakpoint mark and IP index mark

CPU Registers

Output Window

Memory Window with hex dump of memory

Debugger tool-bar

RMB click here

MS VC++ Developer Studio debugger screen
/* demo of assembler within a C prog*/
#include <stdio.h>
#include <stdlib.h>

int main (void)
{
    char format[] = "Hello World\n";  //declare variables in C

    __asm {
        ;switch to inline assembler
        mov ecx,10  ;initialize loop counter
        Lj:        push ecx  ;loop count index saved on stack
                   lea eax,format
                   push eax;address of string, stack parameter
                   call printf ;use library code subroutine
                   add esp,4 ;clean 4 byte parameter off stack
                   pop ecx ;restore loop counter ready for test
                   loop Lj ;dec ECX, jmp back IF NZ
    }

    return 0;
}
1. CPU registers
2. Program memory with labels and disassembled mnemonics
3. Data memory with ASCII decode table
4. Output screen for your program under test
5. Stack, but only for the return addresses.

Debug Toolbar in VC++ Dev Studio
[ESC] open the Start Menu on the Taskbar. You can then open applications on the desktop, this switches between desktop, Taskbar and Start menu. If you already have the Start menu, [Tab] switches between Applications.

Alt [F4] terminate current application
This can also terminate Windows if you are on the desktop!

Alt [Tab] switch to next window
Shift Alt [Tab] switch to preceding window

[ESC] this sometimes cancels the previous action

[F1] display the On-line Help for applications

Shift [F1] context sensitive help

[F2] If an icon is highlighted you can change its name

[F3] get Find

**Keyboard Shortcuts for Windows**
8. CSA - Subroutines

The "where to return to?" problem

The Stack solution
System stack in main memory, SP register in CPU
```c
#include <stdio.h>
#define NCLASS 10

int maths_scores[NCLASS];
int tech_scores[NCLASS];

float average(int x, int * y) {
    int i;
    float av;
    for (i=0; i<x; i++) {
        av += *(y+i);
    }
    return av/x;
}

void main( void) {
    int i;
    ....
    printf("Average of maths = %3.1f\n", average( NCLASS, maths_scores));
    ....
    printf("Average of technology = %3.1f\n", average( NCLASS, tech_scores));
    ....
}
```

Stack growing downwards in memory

Stack operation

PUSH EAX ;push 32bit word in A onto stack
CALL printf ;do something
POP EBX ;pop the 32bit word from stack
ADD ESP,4 ; scrub a longword off the stack
SUB ESP,256 ; open up 256 bytes of space on stack

1. to save the **return address** during PROCEDURE calls
2. to pass **parameters** into PROCEDURES
3. to allocate **Local Variable** storage space (stack frame)
4. as temporary **scratch-pad** storage for register values

**Uses of the system stack**
Disassembled C program with stack operations
Stack growing

Stack area

73 10 40 00  
0A 00 00 00  
30 5A 41 00  

return address  
NCLASS  
maths_scores  

Stack Growing
Subroutine calls with stack frame data
#include <stdio.h>

char* getname(void) {
    char nstring[25];
    printf("Please type your name: ");
    gets(nstring);
    putchar(\n');
    return nstring; //SERIOUS ERROR IN THIS PROGRAM
}

int main(void) {
    char* myname;
    myname = getname();
    printf("%s\n", myname);
    return 0;
}

Spot the error here!

Interrupt Service Routines (ISR) as h/w triggered subroutines

call filter1  
lea esi,filter1
    ...
    ...
    ...
    call [esi]

Late binding

CSA ch 8 - p 98
9. CSA - Simple I/O

1. Dedicated and Periodic polling
2. Interrupt driven
3. Direct Memory Access (DMA)

Different Input/Output Techniques
Software Access to Hardware

Categories of Peripheral chip register

1. Command Registers
2. Status Registers
3. Data Registers
Functional Description

Data Bus Buffer
This three-state bi-directional 8-bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic
The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(CS) Chip Select. A ‘low’ on this input pin enables the communication between the 82C55A and the CPU.

(RD) Read. A ‘low’ on this input pin enables 82C55A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to ‘read from’ the 82C55A.

(WR) Write. A ‘low’ on this input pin enables the CPU to write data or control words into the 82C55A.

(A0 and A1) Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).

RESET Reset. A ‘high’ on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode. ‘Bus hold’ devices internal to the 82C55A will hold the I/O port inputs to a logic ‘1’ state with a maximum hold current of 400mA.

Group A and Group B Controls
The functional configuration of each port is programmed by the systems software. In essence, the CPU “outputs” a control word to the 82C55A. The control word contains information such as “mode”, “bit set”, “bit reset”, etc., that initializes the functional configuration of the 82C55A.

Each of the Control blocks (Group A and Group B) accepts “commands” from the Read/Write Control logic, receives “control words” from the internal data bus and issues the proper commands to its associated ports.

Control Group A - Port A and Port C upper (C7 - C4)
Control Group B - Port B and Port C lower (C3 - C0)

The control word register can be both written and read as shown in the “Basic Operation” table. Figure 4 shows the control word format for both Read and Write operations. When the control word is read, bit D7 will always be a logic ‘1’, as this implies control word mode information.
Mode 0 - basic byte-wide input and output ports
Mode 1 - bytes passed by strobed (asynchronous) handshake
Mode 2 - tri-state bus action

Control Register for the 8255 PIO

Alternative I/O or Memory mapping

// Win-98. Initializes 8255 at 0x1f3: Port A IN; B OUT; C OUT
outp((short)0x1F3, 0x90); // init 8255 cmd reg

Initialization using C


**Accessing Registers in Memory-mapped I/O**

<table>
<thead>
<tr>
<th>Device</th>
<th>Size</th>
<th>Address Pins</th>
<th>Address bus</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROM1</td>
<td>1MB</td>
<td>20</td>
<td>000x ++++ ++++ ++++ ++++ ++++</td>
<td>00 0000 - 0F FFFF</td>
</tr>
<tr>
<td>RAM1</td>
<td>2MB</td>
<td>21</td>
<td>001+ ++++ ++++ ++++ ++++ ++++</td>
<td>20 0000 - 3F FFFF</td>
</tr>
<tr>
<td>RAM2</td>
<td>2MB</td>
<td>21</td>
<td>010+ ++++ ++++ ++++ ++++ ++++</td>
<td>40 0000 - 5F FFFF</td>
</tr>
<tr>
<td>RAM3</td>
<td>2MB</td>
<td>21</td>
<td>011+ ++++ ++++ ++++ ++++ ++++</td>
<td>60 0000 - 7F FFFF</td>
</tr>
<tr>
<td>I/O</td>
<td>4B</td>
<td>2</td>
<td>111x xxxx xxxx xxxx xxxx xx++</td>
<td>E0 0000 - E0 0003</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E0 0004 - E0 0007</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E0 0008 - E0 000B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>E0 000C - E0 000F</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>aliases</td>
</tr>
</tbody>
</table>
**Polled I/O in ASM & C**

```assembly
LOOP:  IN AX,RXSTATUS ;read status port
       TEST AL,RXRDY ;test device status
       JZ LOOP ;if no data go back again

DATIN: IN AX,RXDATA ;get Rx data & clear RXRDY flag
       OR AL,AL ;test for end marker
       JZ COMPLETE ;jmp out if finished
       MOV [DI],AL ;save character in data buffer
       INC DI
       JMP LOOP ;back for more input

COMPLETE: .... ;character string input complete
```

```c
do {
   while (!(*(BYTE*)RXSTATUS & RXRDY)) {} /* wait for data */
} while (*pch++ = *(BYTE*)RXDATA) ; /* check for a NULL */
```
<table>
<thead>
<tr>
<th>Device Description</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>System buses operate</td>
<td>500 Mbyte/sec</td>
</tr>
<tr>
<td>Blocks of characters can be moved</td>
<td>100 Mbyte/sec</td>
</tr>
<tr>
<td>Ethernet transfers data</td>
<td>10 Mbytes/sec</td>
</tr>
<tr>
<td>Telephone call needs</td>
<td>8 Kbyte/sec</td>
</tr>
<tr>
<td>Serial lines frequently run</td>
<td>1 Kbyte/sec</td>
</tr>
<tr>
<td>Epson printers operate</td>
<td>100 byte/sec</td>
</tr>
<tr>
<td>Keyboards send</td>
<td>4 byte/sec</td>
</tr>
</tbody>
</table>

Relative device speeds
/* io.h  68k header file with h/w definitions */

/* messages */
#define PAPER_OUT -1
#define DE_SELECT -2
#define YES 0
#define NO -1
#define OK 0

/* address, offsets and setting for M68681 DUART */
#define DUART 0xFFFF80 /*base address*/
#define ACR 9 /*aux control reg*/
#define CRA 5 /*command reg A*/
#define MRA 1 /*mode reg A*/
#define CSRA 3 /*clock select A*/
#define SRA 3 /*status reg A*/
#define RBA 7 /*rx reg A*/
#define TBA 7 /*tx reg A*/
#define RXRDY 1 /*bit mask for rx ready bit*/
#define TXRDY 4 /*bit mask for tx ready bit*/

/* Settings for the Motorola M68230 Parallel Interface Timer */
These only deal with mode 0.0, and for ports B and C
No details about the timer.*/

/* PI/T offsets and addresses, PIT registers are all on odd addresses */
#define PIT 0xFFFF40 /*address of PI/T*/
#define BCR 0Xf /*offset for port B cntrl Reg*/
#define BDDR 7 /*offset for B data direction*/
#define BDR 0X13 /*offset port B data reg*/
#define CDR 0X19 /*offset port C data reg*/

/* Parallel port settings masks and modes */
#define MODE0 0x20 /* mode 0.0, 2X buff i/p, single buff o/p*/
#define MODE01X 0x80 /* mode 0.1X, unlatch i/p, 1X buff o/p*/
#define OUT 0xFF /* all bits output: 0 - i/p, 1 - o/p*/
#define STROBE_MINUS 0x28 /* strobe printer -ve*/
#define STROBE_PLUS 0x20 /* strobe printer +ve*/
#define PRINT_ST 1 /* paper out pin 00000001*/
#define PAPER_ST 2 /* paper out pin 00000010*/
#define SELECT_ST 4 /* selected pin 00000100*/

CSA ch 09 - p 107
/* Initialization and data transfer for 68k SBC */

#include "io.h"

/* set up Mc68681 DUART serial port A only */
void dinit() {
  register char *p;
  register int i;

  p = (char *)DUART;
  *(p+ACR) = 128; /* set baud rate */
  *(p+CRA) = 16; /* reset Rx */
  *(p+MRA) = 19; /* no modem, no PARITY, 8 bits */
  *(p+MRA) = 7; /* no ECHO, no modem cntrl, 1 STOP */
  *(p+CRA) = 5; /* enable Rx & Tx */
  *(p+CSRA)= 187; /* Rx & Tx at 9600 */

  p = (char *) PIT; /* set to base address of PI/T */
  *(p+BCR ) = MODE0; /* mode 0.0 */
  *(p + BDDR ) = OUT;

  for(i=0; i != 1000;i++) ; /* init delay*/
}

/* set up 68230 PIT for print out on port B */
void pinit() {
  char *p;
  p = (char *)PIT; /* set to base address of PI/T */
  *(p + BCR ) = MODE0; /* mode 0.0 */
  *(p + BDDR ) = OUT;
}

/* get char from serial port A returns character */
char get() {
  register char *p;
  p = (char *)DUART;
  while ( !( *(p+SRA) & RXRDY )) { }; /* block here */
  return *(p+RBA);
}

/* put character c to serial port A */
void put( char c) {
  register char *p;
  p = (char *)DUART;
  while ( !( *(p+SRA) & TXRDY )) { }; /* block here */
  *(p+TBA) = c;
}

/* put string to serial port A using put routine */
void puts(char* p) {
  while( *p )
    put(*p++);
  put('\n');
}

Continues
/* put character to parallel port */
int print(int c) {
    register char * p;
    p = (char *) PIT;
    while ( *(p + CDR) & PRINT_ST ) {
        if ( !( *(p + CDR) & PAPER_ST) )
            return (PAPER_OUT);
        if ( !( *(p + CDR) & SELECT_ST) )
            return (DE_SELECT);
    }
    *(p + BDR) = c; /* send data */
    *(p + BCR) = STROBE_MINUS; /* strobe positive */
    *(p + BCR) = STROBE_PLUS; /* strobe negative */
    return OK;
}
Telephonic Interruptions

Alternative interrupt arrangements
### Part of the PC Interrupt Vector Table (IVT)

<table>
<thead>
<tr>
<th>Int Number</th>
<th>Function</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Integer Divide Error</td>
<td>INT</td>
</tr>
<tr>
<td>1</td>
<td>NMI, Power fail</td>
<td>INT</td>
</tr>
<tr>
<td>2</td>
<td>Single Step Trace</td>
<td>INT</td>
</tr>
<tr>
<td>3</td>
<td>Breakpoint</td>
<td>INT</td>
</tr>
<tr>
<td>4</td>
<td>Numeric Overflow</td>
<td>INT</td>
</tr>
<tr>
<td>5</td>
<td>Screen dump to printer</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>----</td>
<td>INT</td>
</tr>
<tr>
<td>7</td>
<td>System Timer</td>
<td>INT</td>
</tr>
<tr>
<td>8</td>
<td>BIOS/Video</td>
<td>INT</td>
</tr>
<tr>
<td>9</td>
<td>BIOS CHK INT</td>
<td>INT</td>
</tr>
<tr>
<td>10</td>
<td>BIOS Disk INT</td>
<td>INT</td>
</tr>
<tr>
<td>11</td>
<td>BIOS Comms INT</td>
<td>INT</td>
</tr>
<tr>
<td>12</td>
<td>BIOS/MM/INT</td>
<td>INT</td>
</tr>
<tr>
<td>13</td>
<td>BIOS Disk INT</td>
<td>INT</td>
</tr>
<tr>
<td>14</td>
<td>BIOS Disk INT</td>
<td>INT</td>
</tr>
<tr>
<td>15</td>
<td>BIOS KBD INT</td>
<td>INT</td>
</tr>
<tr>
<td>16</td>
<td>BIOS Print INT</td>
<td>INT</td>
</tr>
<tr>
<td>17</td>
<td>BIOS Softboot INT</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>BIOS TOD INT</td>
<td>INT</td>
</tr>
<tr>
<td>0F</td>
<td>LPT1:</td>
<td>IRQ7</td>
</tr>
<tr>
<td>0E</td>
<td>FDC:</td>
<td>IRQ6</td>
</tr>
<tr>
<td>0D</td>
<td>Soundcard</td>
<td>IRQ5</td>
</tr>
<tr>
<td>0C</td>
<td>COM1:</td>
<td>IRQ4</td>
</tr>
<tr>
<td>0B</td>
<td>COM2:</td>
<td>IRQ3</td>
</tr>
<tr>
<td>0A</td>
<td>----</td>
<td>IRQ2</td>
</tr>
<tr>
<td>09</td>
<td>KBD:</td>
<td>IRQ1</td>
</tr>
<tr>
<td>08</td>
<td>System Timer</td>
<td>IRQ0</td>
</tr>
<tr>
<td>07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04</td>
<td></td>
<td></td>
</tr>
<tr>
<td>03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**PIC 1**

- IRQ0
- IRQ1
- IRQ2
- IRQ3
- IRQ4
- IRQ5
- IRQ6
- IRQ7
- IRQ8
- IRQ9
- IRQ10
- IRQ11
- IRQ12
- IRQ13
- IRQ14
- IRQ15

**PIC 2**

- IRQ0
- IRQ1
- IRQ2
- IRQ3
- IRQ4
- IRQ5
- IRQ6
- IRQ7
- IRQ8
- IRQ9
- IRQ10
- IRQ11
- IRQ12
- IRQ13
- IRQ14
- IRQ15

**to CPU interrupt**

---

**Int Function Source**

- **77** Hard Disk2 IRQ15
- **76** Hard Disk1 IRQ14
- **75** 8087 IRQ13
- **74** PS/2 Mouse IRQ12
- **73** Soundcard IRQ11
- **72** Network IRQ10
- **71** Redirected IRQ2
- **70** RTC IRQ8

---

**Part of the PC Interrupt Vector Table (IVT)**

---

**CSA Rob Williams**

**Pearson Education (c) 2006**

---

**CSA ch 09 - p 111**
Displaying PC IRQs using Windows NT

<table>
<thead>
<tr>
<th>IRQ</th>
<th>Device</th>
<th>Bus</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>EDC02ap1</td>
<td>0</td>
<td>isa</td>
</tr>
<tr>
<td>03</td>
<td>Soundcard</td>
<td>0</td>
<td>isa</td>
</tr>
<tr>
<td>04</td>
<td>Serial</td>
<td>0</td>
<td>isa</td>
</tr>
<tr>
<td>05</td>
<td>opGeo</td>
<td>0</td>
<td>isa</td>
</tr>
<tr>
<td>06</td>
<td>Floppy</td>
<td>0</td>
<td>isa</td>
</tr>
<tr>
<td>11</td>
<td>SN19900N</td>
<td>0</td>
<td>isa</td>
</tr>
<tr>
<td>14</td>
<td>olapi</td>
<td>0</td>
<td>isa</td>
</tr>
<tr>
<td>15</td>
<td>atapi</td>
<td>0</td>
<td>isa</td>
</tr>
</tbody>
</table>
Locating the Interrupt Service Routine

1. I/O data transfer request
2. Software TRAP (SVC)
3. Machine Failure
4. Real-time Tick
5. Run-time Software Error
6. System Reset or Watchdog

Possible Sources of Interrupts
Imposing access controls using interrupts

Interrupts per second
Data Variables in Memory

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msecs</td>
<td>990</td>
</tr>
<tr>
<td>secs</td>
<td>59</td>
</tr>
<tr>
<td>mins</td>
<td>59</td>
</tr>
<tr>
<td>hrs</td>
<td>01</td>
</tr>
</tbody>
</table>

Display Routines

1. Display secs
2. Display mins
3. Display hrs

ISR

 interrupt request

msecs++

? msecs = 0

secs++

? secs = 0

mins++

? mins = 0

hrs++

? hrs = 0

rte

Shared data corruption problem

1. disable interrupts
2. serialise the access
3. use a semaphore

Critical region protection
Serialized access to shared data
Operating system managed I/O

Using DMA to Transfer Data

<table>
<thead>
<tr>
<th>Channel</th>
<th>Function</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DRAM refresh</td>
<td>8 bits</td>
</tr>
<tr>
<td>1</td>
<td>SoundBlaster</td>
<td>8 bits</td>
</tr>
<tr>
<td>2</td>
<td>Floppy Drive</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>cascaded to second DMA controller</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>SoundBlaster</td>
<td>16 bits</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Designation of the PC DMA channels

<table>
<thead>
<tr>
<th>Input</th>
<th>Process</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The importance of overlapping operations

```c
#include <stdio.h>

int main(void) {
    int answer;
    do {
        printf("please enter a single letter: ");
        answer = getchar();
        putchar('
');
        printf("%c\n", answer);
    } while (answer != 'E');

    return 0;
}
```

Problems with keyboard input
#include <stdio.h>

int main(void) {
    int answer;
    do {
        printf("please enter a single letter: ");
        answer = getchar();
        getchar();
        putchar('
');
        printf("%c\n", answer);
    } while (answer != 'E');

    return 0;
}

scanf("%c%c", &answer);
10. CSA - Serial Communications

**data** - compression and coding schemes, quantity

**timing** - synchronization of rx with tx: frequency and phase

**signaling** - error handling, flow control, and routing

Three key issues for communication

Receiver must sample near the middle of an incoming bit
Clock drift problems for asynchronous receivers

SYN - special flag Byte to assist receiver with Byte-level synching.
  only used when the channel is operating in Synchronous mode
SOH - Start of a message header
STX - Start of message text block
ETX - End of message text block. Messages can be split into multiple blocks.
EOT - End of message transmission

Parity Bits       - simple to apply, not very secure
Block Checksums- simple to apply, not very helpful
Polynomial Division- more complex, better security

Error Detection and Correction Techniques
Using XOR gates to compute parity

<table>
<thead>
<tr>
<th>Data to be sent</th>
<th>Parity is Computed and appended to make EVEN for transmission</th>
<th>Transmit</th>
<th>New Parity value computed and compared</th>
</tr>
</thead>
<tbody>
<tr>
<td>0110_0111</td>
<td>0110_0111 1</td>
<td>No errors</td>
<td>0110_0111 1</td>
</tr>
<tr>
<td>0111_0110</td>
<td>0111_0110 1</td>
<td>Error</td>
<td>0111_0110 0</td>
</tr>
<tr>
<td>0111_0100</td>
<td>0111_0100 0</td>
<td>Errors</td>
<td>0111_0100 0</td>
</tr>
</tbody>
</table>

Error detection using single appended parity bit
Triple Parity Bit Assignment

Assigning Parity Bits to Longer Words

Parity bits

<table>
<thead>
<tr>
<th>p</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>2^p</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
<td>128</td>
<td>256</td>
</tr>
</tbody>
</table>

Data bits

| d | 0 | 1 | 4 | 11 | 26 | 57 | 120 | 247 |

\[ d = 2^p - (p + 1) \]

Data and parity bits to achieve single error correction

\[
\begin{bmatrix}
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
7 & 6 & 5 & 4 & 3 & 2 & 1 \\
d & d & d & p & d & p & p
\end{bmatrix}
\times
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 0 \\
1 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
= \begin{bmatrix}
0 & 0 & 1
\end{bmatrix}
\]

So p3 = 0, p2 = 0, and p1 = 1

giving \([1\ 0\ 1\ 0\ 1\ 0\ 1]\) for transmission

Calculating a 4d-3p Syndrome (Transmitter)
The 4d-3p syndrome (receiver) with 1 bit error

\[
\begin{bmatrix}
1 & 0 & 0 & 1 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 0 \\
1 & 0 & 1 \\
1 & 0 & 0 \\
0 & 1 & 1 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & 1
\end{bmatrix}
\]

<table>
<thead>
<tr>
<th>Syndrome</th>
<th>No error</th>
<th>Single error</th>
<th>Double error</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>agrees</td>
<td>error</td>
<td>agrees</td>
</tr>
<tr>
<td>Syndrome</td>
<td>0</td>
<td>nonzero</td>
<td>nonzero confused</td>
</tr>
</tbody>
</table>

Single error correction, double error detection by multiple parity
Motorola S-Record format with trailing checksum

Example Fragment of a Motorola S Record Format File

\[ 08 + 01 + 22 + 40 + 4E + 75 + 00 + 00 = 12E \]

*forget the 1 as overflow, leaving 2E ( 0010 1110 )

*invert the bits 1101 0001 ( \textbf{D1} ) \quad \text{THE CHECKSUM !}
Calc. of a CRC at sender and receiver for data item 11001

1. All error bursts of 16 bits or less,
2. All odd numbers of bits in error,
3. 99.998% of all error bursts of any length.

CRC Generation using Shift Registers and XOR gates
Flow control techniques

RS232 flow control techniques

Serial links dedicated route, no addressing needed
LAN broadcast transmission, receiver does the identification
WAN selective routing can be dynamically changed

Data routing methods for serial communications

RS232 voltages representing ASCII ‘1’ (31H)
DCE (9 pin D-type) IBM COM1 Modem Port

- 2 Rx Data
- 3 Tx Data
- 4 DTR Data Terminal Ready
  --- 5 Earth
- 7 RTS Ready to Send
- 8 CTS Clear to Send

RS232 9-way D-type Pin Functions (COM1 & COM2)

Setting COM1 port Parameters with Hyperterminal
Exchanging messages across an RS232 link on a PC
Attaching a UART, serial line interface
/* Filetrans.c */
#include <stdio.h>
#include <conio.h>
define CNTRLZ 0x1A

int main(void) {
    FILE * fp
    FILE * dp;
    int c;
    if ((fp=fopen("C:\TEMP\text.dat", "rt")) ==NULL) {
        printf("fail to open data file\n");
        return 1;
    }
    if ((dp = fopen("COM2", "wt") ) == NULL) {
        printf("fail to open COM port\n");
        return 1;
    }
    while ((c = fgetc(fp)) != EOF) {
        fputc(c, dp);
    }
    fputc(CNTRLZ, dp);
    fflush(dp);
    fclose(fp);
}

/* Filereceive.c */
#include <stdio.h>
#include <conio.h>
define CNTRLZ 0x1A

int main(void) {
    FILE * fp;
    FILE * dp;
    int c;
    if ((fp=fopen("C:\TEMP\text.dat", "w")) ==NULL) {
        printf("fail to open data file\n");
        return 1;
    }
    if ((dp = fopen("COM2", "r") ) == NULL) {
        printf("fail to open COM port\n");
        return 1;
    }
    while ((c= fgetc(dp)) != CNTRLZ) {
        fputc(c, fp);
    }
    fflush(fp);
    fclose(fp);
    return 0;
}

Slow inter-PC file transfers vis COM2
```c
#include <stdio.h>
#include <conio.h>
#include <windows.h>
#include <winbase.h>

HANDLE hCom;
char inpacket[16], outpacket[16];

/// // Initializes PC COM2 port to non-blocking mode
///
/// void initcomm(void)
{
   COMMTIMEOUTS noblock;
   DCB dcb;

   hCom = CreateFile("COM2", GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
   if (hCom == INVALID_HANDLE_VALUE) {
      dwError = GetLastError();
      printf("INVALID_HANDLE_VALUE()");
   }
   fSuccess = GetCommTimeouts(hCom, &noblock);
   noblock.ReadTotalTimeoutConstant = 1;
   noblock.ReadTotalTimeoutMultiplier = MAXDWORD;
   noblock.ReadIntervalTimeout = MAXDWORD;
   fSuccess = SetCommTimeouts(hCom, &noblock);

   fSuccess = GetCommState(hCom, &dcb);
   if (!fSuccess) printf("GetCommState Error!");
   dcb.BaudRate = 9600;
   dcb.ByteSize = 7;
   dcb.fParity = TRUE;
   dcb.Parity = EVENPARITY;
   dcb.StopBits = TWOSTOPBITS;
   dcb.fRtsControl = RTS_CONTROL_HANDSHAKE;
   dcb.fOutxCtsFlow = TRUE;
   fSuccess = SetCommState(hCom, &dcb);
   if (!fSuccess) printf("SetCommState Error!");
   else printf("Comm port set OK!0);
```

Continues
// Reads COM2, single character
// IF !char on COM2 return 0, ELSE return ASCII char
//
char readcomm()
{
    char item;
    int ni;
    fSuccess = ReadFile( hCom,
            &item,
            1,
            &ni,
            NULL
    );
    if (ni >0 ) return item;
    else return 0;
}

// tests and reads keyboard
// IF !char on kbd return 0, ELSE return ASCII char
//
char readkbd()
{
    if (kbhit() ) return _getch();
    else return 0;
}

Using COM2 in Non-blocking Mode
Optical disk direction and speed sensing

Arrangement for a PC Serial Mouse with UART

Optical mouse image sensor DSP
hardware issues
  plugs & sockets don’t fit: 25/9 pin, sockets/pins
  Tx and Rx pins confused - crossed vs uncrossed lead
different plug configurations
  incorrect wiring of h/w flow controls (CTS/RTS)
reversed internal IDC ribbon cables
incorrectly assembled IDC ribbon cables
incorrectly installed interface card (IRQ, dma, port no.)
serial port hardware not initialized

incompatible transmission formats
  ASCII vs EBCDIC or Unicode
  line speed setting: 1200, 2400, 9600 bps
error checks: odd/even/none parity
ASCII char length: 7 vs 8 bits
number of stop bits
user defined packet lengths
CR-LF line terminator differences in files
tab vs multiple SP differences
Word Processor control characters (Word)
EOF problems

flow control failure
  CTS input uncontrolled by receiver
  RTS/CTS talking to XON/XOFF
intermediate buffers on end-to-end flow control
unread echo characters on serial lines
RAM buffer threshold problems

software problems
  sending/receiving data through wrong channel
incorrect device driver installed
uninstalled device driver

Tips and hints on serial connection failure
Control - used by the root hub to pass on configuration instructions and data to the devices, especially used during the initialization period.

Isochronous - timed data transfers for devices with real-time data streams.

Bulk - simple non-time sensitive

Interrupt - USB is not an interrupt system, it depends on timed polling from the hub to pick up data, such as keyboard input.

Universal Serial Bus Connectivity

Intel 8x931 USB Peripheral Microcontroller
Using modems to transfer data on the telephone network

Frequency Modulation Technique

<table>
<thead>
<tr>
<th>ITU Cat</th>
<th>Capacity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>V.21</td>
<td>300/600 bps</td>
<td>Frequency shift</td>
</tr>
<tr>
<td>V.22</td>
<td>1200 bps</td>
<td>Phase shift</td>
</tr>
<tr>
<td>V.22bis</td>
<td>2400 bps</td>
<td>Amplitude &amp; Phase shift</td>
</tr>
<tr>
<td>V.29</td>
<td>9600 bps</td>
<td>Phase shift</td>
</tr>
<tr>
<td>V.32</td>
<td>9600 bps</td>
<td>Amplitude &amp; Phase shift</td>
</tr>
<tr>
<td>V.32bis</td>
<td>14.4 kbps</td>
<td>Amplitude &amp; Phase shift</td>
</tr>
<tr>
<td>V.17</td>
<td>14.4 bps Fax</td>
<td>Amplitude &amp; Phase shift</td>
</tr>
<tr>
<td>V.34</td>
<td>28.8 kbps</td>
<td>Amplitude &amp; Phase shift</td>
</tr>
</tbody>
</table>

Modem Standards and Coding Schemes
<table>
<thead>
<tr>
<th>Command</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATA</td>
<td>Answer incoming call</td>
</tr>
<tr>
<td>ATDnnn-nnnn</td>
<td>Tone dials the phone number nnn-nnnn</td>
</tr>
<tr>
<td>ATL</td>
<td>Redials last number dialed</td>
</tr>
<tr>
<td>ATPDnnn-nnnn</td>
<td>Pulse dial nnn-nnnn</td>
</tr>
<tr>
<td>ATW</td>
<td>Wait for dial tone</td>
</tr>
<tr>
<td>ATH0</td>
<td>Hang up</td>
</tr>
<tr>
<td>ATM0</td>
<td>Speaker off</td>
</tr>
<tr>
<td>ATM1</td>
<td>Speaker is on until a carrier is detected</td>
</tr>
<tr>
<td>ATM2</td>
<td>Speaker is always on</td>
</tr>
<tr>
<td>ATO0</td>
<td>Puts modem in data mode</td>
</tr>
<tr>
<td>ATO1</td>
<td>Takes modem out of data mode</td>
</tr>
<tr>
<td>ATY0</td>
<td>Disable disconnection on pause</td>
</tr>
<tr>
<td>ATY1</td>
<td>Enable disconnection on pause</td>
</tr>
</tbody>
</table>

Some of the Hayes Modem AT Command Set

Phase Modulation Increases the Bit Signalling Rate
Phase shift modem with a single carrier frequency signalling 0 or 1 with 0 or $\pi$ phase shift

Quad Phase, Single Amplitude
$\pi/4, -\pi/4, 3\pi/4, -3\pi/4$
4 level QAM, 2B1Q

Oct Phase, Dual Amplitude
$0, \pi/4, \pi/2, -\pi/4, -\pi/2, 3\pi/4, -3\pi/4, \pi$
8 level QAM, 3B1O

Oct Phase, Quad Amplitude,
32 level QAM
V32 modems

Amplitude Phase Diagrams Illustrating some Modulation Schemes
11. CSA - Parallel connections

<table>
<thead>
<tr>
<th></th>
<th>Standard Parallel Port</th>
<th>100 kbytes/sec</th>
<th>Output software operated</th>
</tr>
</thead>
<tbody>
<tr>
<td>EPP</td>
<td>Enhanced Parallel Port</td>
<td>1 Mbytes/sec</td>
<td>Input/Output h/w handshake circuits</td>
</tr>
<tr>
<td>ECP</td>
<td>Extended Capability Port</td>
<td>5 MBytes/sec</td>
<td>Input/Output DMA with FIFO</td>
</tr>
</tbody>
</table>

**PC Parallel Port (Centronics) Standards**

<table>
<thead>
<tr>
<th>Pin D-25</th>
<th>SPP</th>
<th>Host</th>
<th>Printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strobe</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>data bit 0</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>data bit 1</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>data bit 2</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>data bit 3</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>data bit 4</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>data bit 5</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>data bit 6</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>data bit 7</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>ACK</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>BUSY</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>PE Paper Out</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>SLCT</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>auto LF</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Error</td>
<td>←</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>INIT</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>SLCT IN</td>
<td>→</td>
<td></td>
</tr>
<tr>
<td>18-25</td>
<td>GRND</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**The Centronics Standard Interface (SPP)**

**Sequence of Events within a Centronics Data Transfer**

CSA Rob Williams
Pearson Education (c) 2006
The Centronics Enhanced Interface (EPP / ECP)

1. fitting a large memory buffer inside the printer
2. run a background print spooler
3. use a full multi-tasking system

How to prevent delays caused by a slow printer
Small Computer Systems Interface (SCSI) and Command Packet

**BSY** - Busy indicates that someone is currently using the bus.

**SEL** - Select allows the initiator to select a target and by the target to resume an interrupted session.

**C/D** - Control / Data is controlled by the target to indicate whether control or data items are being transferred on the data bus.

**I/O** - Input / Output allows the target to define the direction of the data transfer.

**ATN** - Attention is used by the master to tell the slave that data is available on the bus.

**MSG** - Message, activated by the target during the message phase of transfer.

**REQ** - Request, used by the target device, signals to the master that data can be transmitted. It is part of the REQ / ACK handshake pair.

**ACK** - Acknowledge, controlled by the initiator to confirm a transfer.

**RST** - Reset bus, forces all attached devices to stop activity and reset the hardware.
<table>
<thead>
<tr>
<th>Group 1</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Test unit ready</td>
<td>13 Verify</td>
</tr>
<tr>
<td>01</td>
<td>Rezero unit</td>
<td>14 Recover buffer</td>
</tr>
<tr>
<td>03</td>
<td>Request sense</td>
<td>15 Mode select</td>
</tr>
<tr>
<td>04</td>
<td>Format unit</td>
<td>16 Reserved unit</td>
</tr>
<tr>
<td>05</td>
<td>Read block limits</td>
<td>17 Release unit</td>
</tr>
<tr>
<td>07</td>
<td>Reassign blocks</td>
<td>18 Copy</td>
</tr>
<tr>
<td>08</td>
<td>Read</td>
<td>19 Erase</td>
</tr>
<tr>
<td>0A</td>
<td>Write</td>
<td>1A Mode sense</td>
</tr>
<tr>
<td>0B</td>
<td>Seek</td>
<td>1B Start / stop</td>
</tr>
<tr>
<td>0F</td>
<td>Read reverse</td>
<td>1C Receive diagnostic</td>
</tr>
<tr>
<td>10</td>
<td>Write file mark</td>
<td>1D Send diagnostic</td>
</tr>
<tr>
<td>11</td>
<td>Space</td>
<td>1E Lock media</td>
</tr>
<tr>
<td>12</td>
<td>Inquiry</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Group 2</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>Read capacity</td>
<td>30 Search data high</td>
</tr>
<tr>
<td>26</td>
<td>Extend addr rd</td>
<td>31 Search data equal</td>
</tr>
<tr>
<td>2A</td>
<td>Extend addr wr</td>
<td>32 Search data low</td>
</tr>
<tr>
<td>2E</td>
<td>Write 7 verify</td>
<td>33 Set limits</td>
</tr>
<tr>
<td>2F</td>
<td>Verify</td>
<td>39 Compare</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3A Copy &amp; verify</td>
</tr>
</tbody>
</table>

**SCSI Message Codes**

![SCSI message codes diagram](image_url)

**An 8 bit PC/ISA-bus Printer Interface Card**

CSA Rob Williams
Pearson Education (c) 2006  
CSA ch 11 - p 145
An extended 16 bit PC/ISA-bus Parallel I/O Card

A comparison of ISA Bus and PC/104 connectors
The PCI bus can operate in two modes:

**Multiplexed Mode** A single 32 bit bus is shared by address and data information. This increases the effective bus width, but reduces the data rate.

**Burst Mode** This is the same trick that EDO DRAM employs. After an address has been sent, several data items will follow in quick succession. The bridge is capable of assembling "packets" of data and bursting it through to the PCI bus when ready.

---

**Relationship of the PCI Bridge to Main Bus**

**The PCI Bridge**
<table>
<thead>
<tr>
<th>Card</th>
<th>Serial Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

Plug ’n Play Sequence

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>MID</th>
<th>PID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptec</td>
<td>9004</td>
<td>36868</td>
</tr>
<tr>
<td>Compaq</td>
<td>1032</td>
<td>4146</td>
</tr>
<tr>
<td>Creative</td>
<td>10F6</td>
<td>4342</td>
</tr>
<tr>
<td>Cyrix</td>
<td>1078</td>
<td>4216</td>
</tr>
<tr>
<td>Epson</td>
<td>1008</td>
<td>4104</td>
</tr>
<tr>
<td>HP</td>
<td>103C</td>
<td>4156</td>
</tr>
<tr>
<td>Intel</td>
<td>8086</td>
<td>32902</td>
</tr>
<tr>
<td>Matsushita</td>
<td>10F7</td>
<td>4343</td>
</tr>
<tr>
<td>Mitsubishi</td>
<td>1067</td>
<td>4199</td>
</tr>
<tr>
<td>Motorola</td>
<td>1057</td>
<td>4183</td>
</tr>
<tr>
<td>NCR</td>
<td>1000</td>
<td>4096</td>
</tr>
<tr>
<td>Toshiba</td>
<td>102F</td>
<td>4143</td>
</tr>
<tr>
<td>Tseng Labs</td>
<td>100C</td>
<td>4108</td>
</tr>
</tbody>
</table>

Example Plug and Play Identity Numbers
PCMCIA Interface
12. CSA - Memory hierarchy

Fully populated main memory

Memory Performance Hierarchy
<table>
<thead>
<tr>
<th>Facility</th>
<th>Size</th>
<th>device</th>
<th>Unit cost, £</th>
<th>£ / Mbyte</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM</td>
<td>64 MB</td>
<td>SDRAM 168 pin DIMM</td>
<td>40</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>32 MB</td>
<td>EDO 72 pin SIMM</td>
<td>50</td>
<td>1.6</td>
</tr>
<tr>
<td>SRAM</td>
<td>256 KB/10ns</td>
<td>SRAM chip</td>
<td>2</td>
<td>64</td>
</tr>
<tr>
<td>SCSI</td>
<td>PCI card</td>
<td></td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>Hard Disk</td>
<td>10GB</td>
<td>IDE</td>
<td>110</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td>10GB</td>
<td>SCSI</td>
<td>250</td>
<td>0.025</td>
</tr>
<tr>
<td>CD-ROM</td>
<td>650 MB</td>
<td>32x IDE</td>
<td>60</td>
<td>0.12</td>
</tr>
<tr>
<td>CD-RW</td>
<td>650 MB</td>
<td>2x IDE</td>
<td>200</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>WORM</td>
<td>disk</td>
<td>0.75</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RW</td>
<td>disk</td>
<td>2.50</td>
<td></td>
</tr>
<tr>
<td>Jazz</td>
<td>1 GB</td>
<td>drive</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 GB</td>
<td>disk</td>
<td>65</td>
<td>0.25</td>
</tr>
<tr>
<td>Zip</td>
<td>100 MB</td>
<td>drive</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100 MB</td>
<td>disk</td>
<td>15</td>
<td>0.75</td>
</tr>
<tr>
<td>DAT</td>
<td>4 GB</td>
<td>SCSI drive</td>
<td>350</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 GB</td>
<td>tape</td>
<td>2.5</td>
<td>0.09</td>
</tr>
<tr>
<td>Floppy</td>
<td>1.4 MB</td>
<td>drive</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1.4 MB</td>
<td>disk</td>
<td>0.5</td>
<td>11</td>
</tr>
</tbody>
</table>

**Memory costs**

Memory access plot, showing locality effects
Array indexing with memory layout of array data
```c
#include <stdio.h>
#include <sys/times.h>
#include <limits.h>
#define MAX 1000

clock_t times(struct tms* b);

main () {
int i, j;
int big[MAX][MAX];
int sum, start, middle, end;
struct tms tbuff;
times( &tbuff); start = tbuff.tms_utime;
for(i=0; i<MAX; i++) {
    for(j=0; j<MAX; j++) {
        sum += big[i][j]; /* <------------ i, j here */
    }
};
times( &tbuff); middle = tbuff.tms_utime;
for(i=0; i<MAX; i++) {
    for(j=0; j<MAX; j++) {
        sum += big[j][i]; /* <------------ j, i here */
    }
};
times( &tbuff); end = tbuff.tms_utime;
printf("First run time is %d \n",(middle - start)*1000/CLK_TCK);
printf("Second run time is %d \n",(end - middle)*1000/CLK_TCK);
}

Demonstrating cache action

rob [80] cc cache.c -o cache
rob [81] cache
First run time is 150
Second run time is 310
rob [82]

Results from the Cache Test

Estimate: 21 x 5ns x 10^6 + 13 x 5ns x 10^3 = 105ms

CSA Rob Williams
Pearson Education (c) 2006
```
Alternative access patterns (strides) for a 2-D array
for(i=0; i<MAX; i++) {
    mov0,%l0
    st%l0,[%fp-8]
    ld[%fp-8],%l0
    cmp%l0,1000
    bge.L118
    nop
    ! block 2
}
.L119:
.L116:
}
for(j=0; j<MAX; j++) {
    mov0,%l0
    st%l0,[%fp-12]
    ld[%fp-12],%l0
    cmp%l0,1000
    bge.L122
    nop
    ! block 3
}
.L123:
.L120:
}
sum += big[i][j];
sethihi(-4000016),%l0
or%l0,%lo(-4000016),%l0
ld(%fp+10),%l1
sll%l0,12,%l2
sll%l0,5,%l1
sub%l2,%l1,%l1
add%l1,%l1,%l2
ld(%fp-12),%l0
sll%l0,2,%l1
add%l2,%l1,%l0
sethihi(-4000016),%l0
or%l0,%lo(-4000016),%l0
stl1,[%fp+10]
}
.L124:
.L122:
} ;
ld(%fp-12),%l0
addl10,1,%l0
stl0, [ % fp-12]
ld(%fp-12),%l0
cmp%l0,1000
b.l.L120
nop
! block 4
).
.L125:
.L118:
1. Folded address space, also known as Direct Mapping
2. Associative (content addressable) memory
3. Hashed mapping

Mapping Addresses from Main to Cache Memory
Address Folding for Direct Mapped Cache
Checking the Address in Associative Memory
1. on start-up of a new program
2. when the cache is too small to hold the active execution set
3. cache line conflict in a direct mapped cache.

Causes of Cache Misses
Virtual memory scheme for main memory overflow

Virtual memory logical page into physical frame address translation
Alternative positions for the memory management unit

The relation between different address designations
Environmental obstacles with R/W disk heads

Schematic Diagram of Hard Disk Unit

\[
\text{drive\_capacity} = \text{no\_of\_surfaces} \times \text{no\_of\_tracks} \times \text{no\_of\_sectors} \times \text{size\_of\_sector}
\]
A computer system having 2 Mbytes of RAM has hard disks with the following characteristics:

- Rotational speed: 3600 rpm
- Track capacity: 16384 bytes
- Heads/cylinder: 10
- Head movement time, track to track: 20 ms
- Average seek time: 50 ms

How long does it take to dump memory onto disk? Assume the disk is empty.

3600rpm = 60 rps
rotational period = 1/60 secs = 1000/60 msec = 16.66 msec = 17 msec
latency = 8.5 msec

data rate = 16 Kbytes / 17 msec = 1 Mbyte/sec
flow time for 2 Mbyte = 2000 msec = 2 sec

2 Mbyte needs 128 tracks or 12.8 cylinders ie 13 head movements
tot time = head movement (seeks) + rotnl delays (latencies) + data flow time
= 1 x 50 + 12 X 20 + 13 X 8.5 + 2000
= 2314 msec

Estimating Hard Disk Data Retrieval Time

<table>
<thead>
<tr>
<th>Disk Performance Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Track to Track Seek</td>
</tr>
<tr>
<td>Average Seek</td>
</tr>
<tr>
<td>Maximum Seek</td>
</tr>
<tr>
<td>Average Latency</td>
</tr>
<tr>
<td>Rotation</td>
</tr>
<tr>
<td>Controller overhead</td>
</tr>
<tr>
<td>Start time</td>
</tr>
<tr>
<td>Computer interface rate</td>
</tr>
<tr>
<td>Media read/write rate</td>
</tr>
<tr>
<td>Sectors per track</td>
</tr>
<tr>
<td>Cylinders (tracks per surface)</td>
</tr>
<tr>
<td>Bytes per sector</td>
</tr>
<tr>
<td>Data zones per surface</td>
</tr>
<tr>
<td>Integrated buffer size</td>
</tr>
<tr>
<td>Memory type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>90650U2</th>
<th>90845U</th>
<th>91020U3</th>
<th>91360U4</th>
<th>92040U6</th>
<th>92040U8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capacity (Gbytes)</td>
<td>6.5</td>
<td>8.45</td>
<td>10.21</td>
<td>13.61</td>
<td>20.42</td>
<td>27.23</td>
</tr>
<tr>
<td>Heads</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Disks</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

A Specification Table for Maxtor Hard Disks
Disk Access Scheduling Techniques

Comparison of Disk Scheduling Techniques

CD data Spiral and Magnetic Disk Concentric Tracks
lacquer coating (30μm)
Polycarbonate disk (1.2mm)
metallic layer

Optical disk read head

beam splitter
light detector

Laser emitter
focusing
tracking

Paper Label
Lacquer
Reflective coating
Upper dielectric
Recording layer
Lower dielectric
Polycarbonate Disk
Hard Coating

Track grooves

1.2 mm

CD-RW disk structure
Discrete Cosine Transform (DCT) base functions as used with MPEG image compression
13. CSA - Programmer’s Viewpoint

Different jobs, different viewpoints

the switch is closed can the valve open

FABWrite(num, B_8255); printf("switch
move EAX, 04H lea EBX,string call pr
92 E6 EA 0F C2 66 3F A1 92 E6 EA 0F C2
1001 0010 1110 0110 1110 1010 0000 1111

Same device, many different viewpoints
Graphical representation of a directory designed for application users

Traditional text-only Unix directory listing using ls

CSA Rob Williams
Pearson Education (c) 2006
CSA ch 13 - p 171
A Unix process pipeline to format and print text used by systems administrators

```bash
rob@milly [80] /etc/mknod pipe1 p
rob@milly [80] /etc/mknod pipe2 p
rob@milly [80] /etc/mknod pipe3 p
rob@milly [81] ls -al pipe*

prw------- 1 rob csstaff 0 Oct 14 18:39 pipe1
prw------- 1 rob csstaff 0 Oct 14 18:39 pipe2
prw------- 1 rob csstaff 0 Oct 14 18:39 pipe3

rob@milly [82] cat letter.tmp >! pipe1 &
rob@milly [82] cat pipe1 >! pipe2 &
rob@milly [82] cat pipe2 >! pipe3 &
```

Demonstrating Unix named pipes
#!/bin/sh
#
# Script converts sar data into graphs - PJN 20/10/1998
#
# Extend the PATH to include gnuplot
PATH=${PATH}:/usr/local/bin ; export PATH

# Procedure to remove non-data lines from log file.
remclutter() {
  grep : |grep -v free |grep -v % |grep -v / |grep -v restarts
}

# Procedure to pad numbers with zeroes to 2 digits.
padnum() {
  NUM=$1
  while [ `/(bin/echo "$\{NUM\}c" | wc -c` -lt 2 ]; do
    NUM="0$\{NUM\}"
  done
  echo $NUM
}

# Procedure to convert time of day timestamps to decimal days.
parsetimes() {
  DAY=0
  OLDHOUR=23
  while read TIME DATA; do
    if [ "$DATA" = "" ]; then
      DATA="0 0 0 0 0 0 0 0 0 0 0 0"
    fi
    HOUR=`echo $TIME | cut -f1 -d:`
    MIN=`echo $TIME | cut -f2 -d:`
    if [ $HOUR -lt $OLDHOUR -a "$MIN" = "00" ]; then
      DAY=`expr $DAY + 1`
    fi
    PTIME=`expr \( (\( $HOUR \* 60 \) + $MIN \) \* 100 \) /1440`
    PTIME=${DAY}.'padnum $PTIME'
    echo "$PTIME $DATA"
    OLDHOUR=$HOUR
  done
}

# Procedure to get data from a named column.
getcol() {
  tr -s '' ' ' | cut -f1,$\{1\} -d\" | tr "'" ''
}
# Determine the i/p and o/p files (for last week’s data).
WEEK=`date +%W`
WEEK=`expr $WEEK - 1`
if [ $WEEK -eq -1 ]; then
   WEEK=52
fi
WEEK=`padnum $WEEK`
DATAFILE=/var/adm/sa/sa$WEEK
OUTFILE=/tmp/$$.graphs

# Process virtual memory data from sar log.
echo "VM usage"
sar -f $DATAFILE -r > /tmp/$$.sar
cat /tmp/$$.sar | remclutter | parsetimes > /tmp/$$.sar-f
rm /tmp/$$.sar

cat /tmp/$$.sar-f | getcol 2 > /tmp/$$.freemem
cat /tmp/$$.sar-f | getcol 3 > /tmp/$$.freeswap

(cat << EOF
set term postscript
set time
set xtic 0,0.5
set title "`hostname` virtual memory usage"
f(x) = (x * 512 ) / 1048576
g(x) = (x * `pagesize`) / 1048576
plot [0:7] []
"/tmp/$$.freemem" thru g(x) title "Free RAM MB" with lines,
"/tmp/$$.freeswap" thru f(x) title "Free Swap MB" with lines
EOF
) | gnuplot > $OUTFILE
rm /tmp/$$.freemem /tmp/$$.freeswap /tmp/$$.sar-f

lp -d ps $OUTFILE
sleep 60; rm $OUTFILE
exit
Example HLL algorithm - Bubble Sort
for the software engineer
void doitall(void) {
  doA();
  doB();
  doC();
}

for (i=0; i<10; i++) {
  doD();
}

x = 0;
while (x < 10) {
  doD();
  x++;
}

if (x > 0) {
  doE();
} else {
  doF();
}

Structure Chart representations of SEQ, IT & SEL

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOOL</td>
<td>unsigned 1 bit value</td>
</tr>
<tr>
<td>char</td>
<td>unsigned 8 bit value</td>
</tr>
<tr>
<td>WCHAR</td>
<td>unsigned 16 bit value</td>
</tr>
<tr>
<td>BYTE</td>
<td>unsigned 8 bit integer</td>
</tr>
<tr>
<td>short</td>
<td>signed 16 bit integer</td>
</tr>
<tr>
<td>WORD</td>
<td>unsigned 16 bit integer</td>
</tr>
<tr>
<td>int</td>
<td>signed 32 bit integer</td>
</tr>
<tr>
<td>LONG</td>
<td>signed 32 bit integer</td>
</tr>
<tr>
<td>unsigned</td>
<td>unsigned 32 bit integer</td>
</tr>
<tr>
<td>DWORD</td>
<td>unsigned 32 bit integer</td>
</tr>
<tr>
<td>float</td>
<td>IEEE 32 bit real</td>
</tr>
<tr>
<td>double</td>
<td>IEEE 64 bit real</td>
</tr>
</tbody>
</table>

Data types for C/C++, popular languages with systems programmers
Flow Chart representation of SEQ, IT & SEL

SEQ
- do A
- do B
- do C

IT
- y
- n
- do D

SEL
- y
- n
- do E
- do F

CALL doA
CALL doB
CALL doC

L1: JZ L2
.
.
.
CALL doD
JMP L1

L2:
.
.
.
MOV CX,10

L3:
.
.
.
CALL doD
LOOP L3

CMP EAX,12
JGE L4
CALL doE
JMP L5

L4 CALL doF
L5
Circuit schematic diagram for the electronic engineer

A multi-level computer functional interaction
Translation activities of an assembler

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>opcode</td>
<td>$FF</td>
</tr>
<tr>
<td>ADDA</td>
<td>opcode</td>
<td></td>
</tr>
<tr>
<td>SUB</td>
<td>opcode</td>
<td></td>
</tr>
<tr>
<td>MOVE</td>
<td>opcode</td>
<td></td>
</tr>
<tr>
<td>start</td>
<td>defined</td>
<td></td>
</tr>
<tr>
<td>exit</td>
<td>undefin</td>
<td></td>
</tr>
<tr>
<td>loop1</td>
<td>defined</td>
<td></td>
</tr>
<tr>
<td>spx</td>
<td>defined</td>
<td></td>
</tr>
<tr>
<td>sprite</td>
<td>defined</td>
<td></td>
</tr>
</tbody>
</table>
Stages of Compilation
CSA Ch 14

- Computers
  - Fetch-execute cycle
  - Hardware
    - CPU
      - Arithmetic Logic Unit
      - Control Unit
      - RISC features
      - ARM processor
      - Pentium
      - Itanium
    - Input-output
      - Parallel communication
      - Serial communication
    - Networking
      - Local Area Networks
        - Ethernet
        - USB
      - Wide Area Networks
        - Other Networks
      - Point to point
      - Visual output
    - Memory
      - Memory hierarchy
      - Cache and main memory
      - Disk filing
    - Parallel processing
  - Software
    - Operating systems
      - Unix
      - MS Windows
    - Tools
      - Compilers and assemblers
      - Subroutines and stacks
      - WIMPs
  - Users' viewpoints
    - Hardware engineer
    - HLL programmer
    - Systems administrator
    - Systems programmer
14. CSA - Local Area Networks

Evolution of computing provision

Displaying LAN traffic using Sun’s perfmeter
Some of the IEEE 802 standards committees

Traditional office bus LAN facility

Star topology, switched hub, ethernet

Star topology with hierarchical hubs
Connections to an RJ45 LAN Plug

A network interface card with 10Base2 & 10BaseT connectors
H/w and s/w layers to manage the LAN Interface
Manchester Encoding for Data and Clock

rising edge - 0,  dropping edge - 1

10101010 10101010 10101010 10101010 10101010 10101010 10101010 10101010

Flag Bytes

10101010 10101010 10101010 10101010 10101011 [Ethernet_packet_body....]
<table>
<thead>
<tr>
<th>Ethernet Standards</th>
<th>Speed</th>
<th>Segment Length</th>
<th>Accessories/Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>10Base5</td>
<td>10Mbps</td>
<td>500m</td>
<td>Thick Ethernet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>minimum tap separation 2.5m, maximum of 4 repeaters, 50 Ω coax cable, vampire tap (Media Access Unit)</td>
</tr>
<tr>
<td>10Base2</td>
<td>10Mbps</td>
<td>200m (165m)</td>
<td>Thin Ethernet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>minimum tap separation 0.5m, maximum of 4 repeaters, 70 Ω coax cable, BNC T-piece bayonet connection</td>
</tr>
<tr>
<td>10BaseT</td>
<td>10Mbps</td>
<td>100m</td>
<td>Switched Ethernet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>end-to-end, simplex, 100 Ω AWG24 twisted pairs cable, RJ45 telecom jack</td>
</tr>
<tr>
<td>100BaseT</td>
<td>100Mbps</td>
<td>205m</td>
<td>Fibre Ethernet</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>end-to-end, simplex, 100 Ω AWG24 twisted pairs cable</td>
</tr>
<tr>
<td>100BaseF</td>
<td>100Mbps</td>
<td>2000m</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>end-to-end, simplex, optic fibres</td>
</tr>
</tbody>
</table>

Various ethernet media standards
\[ T_{\text{packet}} = \frac{500 \times 5 \times 2}{1 \times 10^8} = 50 \mu s \]

\[ t_{\text{bit}} = 0.1 \mu s \]

\[ N_{\text{packet}} = \frac{50}{0.1} = 500 \text{ bits} \]

\[ N_{\text{Bytes}} = \frac{500}{8} = 62.5 \sim 64 \text{ Bytes} \]

**Collision Detection and Transit Times for Ethernet**
<table>
<thead>
<tr>
<th>Binary</th>
<th>Trinary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000_0000</td>
<td>+00+-T</td>
</tr>
<tr>
<td>0000_0001</td>
<td>+0+-+-T</td>
</tr>
<tr>
<td>0000_0010</td>
<td>+0+-0T</td>
</tr>
<tr>
<td>0000_0011</td>
<td>+0++-T</td>
</tr>
<tr>
<td>0000_0100</td>
<td>0+0+-T</td>
</tr>
<tr>
<td>0000_0101</td>
<td>0+-0+T</td>
</tr>
<tr>
<td>0000_0110</td>
<td>++0+0+T</td>
</tr>
<tr>
<td>0000_0111</td>
<td>-0+-0+T</td>
</tr>
<tr>
<td>0000_1000</td>
<td>--00+-T</td>
</tr>
<tr>
<td>0000_1001</td>
<td>0++-0T</td>
</tr>
<tr>
<td>0000_1010</td>
<td>-+0+-0T</td>
</tr>
<tr>
<td>0000_1011</td>
<td>0+-0+T</td>
</tr>
<tr>
<td>0000_1100</td>
<td>+0-0+T</td>
</tr>
<tr>
<td>0000_1101</td>
<td>0+-0+T</td>
</tr>
<tr>
<td>0000_1110</td>
<td>+0-0+T</td>
</tr>
<tr>
<td>0000_1111</td>
<td>+0-0+T</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
</tr>
<tr>
<td>1111_1111</td>
<td>+0+-00T</td>
</tr>
</tbody>
</table>

**8B6T Coding**
### Internal Structure of an Ethernet Data Packet

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-0</td>
<td>Sync Preamble (56 bits)</td>
</tr>
<tr>
<td></td>
<td>SFD</td>
</tr>
<tr>
<td></td>
<td>Destination Address (48 bits)</td>
</tr>
<tr>
<td></td>
<td>Source Address (48 bits)</td>
</tr>
<tr>
<td></td>
<td>Length of Data Field</td>
</tr>
<tr>
<td>0-15</td>
<td>Data 0-1500</td>
</tr>
<tr>
<td></td>
<td>Padding 0-46</td>
</tr>
<tr>
<td></td>
<td>CRC</td>
</tr>
</tbody>
</table>

**error detection**
The 5 Forms of IP v4 numbers and their ranges

 ARP table, translating IP addresses into MAC numbers
Host table, translating acronym into IP addresses

A Layered description of networking software
Inspecting the state of a Unix file systems using df
Installing a virtual drive using Windows XP

Interconnecting LANs using a gateway
Socket communication between remote processes
Communication with Client-Server connection-based (STREAM) sockets
Win32 Socket Function Calls

SOCKET socket(int af, int typesock, int protocol)
int bind(SOCKET mysock, const struct sockaddr *psock, int nlength)
int listen(SOCKET mysock, int qmax)
int connect(SOCKET yoursock, const struct sockaddr *sname, int nlength)
SOCKET accept(SOCKET mysock, struct sockaddr *psock, int *addrlen)
int send(SOCKET yoursock, const char *pdbuff, int dblen, int flags)
int recv(SOCKET mysock, char *pdbuff, int dblen)
int closesocket(SOCKET mysock)
Communication with Client - Server connection-less (DGRAM) sockets
WANs give long distance interconnection for LANs

TCP/IP - are the essential protocols for the Internet
The TCP/IP Stack at Work

<table>
<thead>
<tr>
<th>Number</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>02</td>
<td>ICMP</td>
</tr>
<tr>
<td>06</td>
<td>TCP</td>
</tr>
<tr>
<td>17</td>
<td>UDP</td>
</tr>
</tbody>
</table>

IP Protocol Field Values
TCP port numbers and their services from /etc/services

> cat /etc/services

tcpmux 1/tcp
echo 7/tcp
echo 7/udp
discard 9/tcp sink null
discard 9/udp sink null
systat 11/tcp users
ftp-data 20/tcp
ftp 21/tcp
telnet 23/tcp
smtp 25/tcp mail
time 37/udp timserver
name 42/udp nameserver
whois 43/tcp nicname # usually to sri-nic
gopher 70/tcp # Internet Gopher
finger 79/tcp
www 80/tcp http # World Wide Web
www 80/udp
hostnames 101/tcp hostname # usually to sri-nic
sunrpc 111/udp rpcbind
sunrpc 111/tcp rpcbind
Ethernet, IP, TCP encapsulation
Flow control using a four packet buffer

Differentiating Repeaters, Bridges, and Routers
```
rob@olveston [20] cat /etc/hosts
# Internet host table
#
127.0.0.1  localhost
164.11.253.47  olveston  loghost
164.11.8.16  egg ns0
164.11.253.2  sister ns1
164.11.8.99  ada ns2
164.11.10.5  riff ns3
rob@olveston [21]
rob@olveston [21] netstat -rn
Routing Table:

<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Flags</th>
<th>Ref</th>
<th>Use</th>
<th>Interf</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>UH</td>
<td>0</td>
<td>503</td>
<td>lo0</td>
</tr>
<tr>
<td>164.11.253.0</td>
<td>164.11.253.47</td>
<td>U</td>
<td>3</td>
<td>228</td>
<td>hme0</td>
</tr>
<tr>
<td>224.0.0.0</td>
<td>164.11.253.47</td>
<td>U</td>
<td>3</td>
<td>0</td>
<td>hme0</td>
</tr>
<tr>
<td>default</td>
<td>164.11.253.1</td>
<td>UG</td>
<td>0</td>
<td>14297</td>
<td></td>
</tr>
</tbody>
</table>

Unix netstat utility showing the routing table
```
Traffic flooding without routing decisions

<table>
<thead>
<tr>
<th>RIP Operation</th>
<th>IP Number</th>
<th># router hops</th>
<th># ticks (56ms)</th>
<th>IP Number</th>
<th># router hops</th>
<th># ticks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1st hop</td>
<td></td>
<td></td>
<td>2nd hop</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

RIP packet fields

<table>
<thead>
<tr>
<th>Region</th>
<th>IP Numbers Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>Europe</td>
<td>194.000.000.000 - 195.255.255.255</td>
</tr>
<tr>
<td>N America</td>
<td>198.000.000.000 - 199.255.255.255</td>
</tr>
<tr>
<td>S America</td>
<td>200.000.000.000 - 201.255.255.255</td>
</tr>
<tr>
<td>Pacific Asia</td>
<td>202.000.000.000 - 203.255.255.255</td>
</tr>
</tbody>
</table>

CIDR IP number allocations

Identifier translation required for transmitters
Inspecting the local hosts file

Hierarchical Domain Naming Structure (DNS)
Using the DNS name look-up facility
Netscape Navigator Web Browser

URL = Protocol identifier/Machine name/file path
Introducing world.html to Netscape on Unix

<table>
<thead>
<tr>
<th>Tags</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;HTML&gt;... &lt;/HTML&gt;</td>
<td>Page delimiters</td>
</tr>
<tr>
<td>&lt;HEAD&gt;... &lt;/HEAD&gt;</td>
<td>Page heading</td>
</tr>
<tr>
<td>&lt;TITLE&gt;... &lt;/TITLE&gt;</td>
<td>http title (invis)</td>
</tr>
<tr>
<td>&lt;BODY&gt;... &lt;/BODY&gt;</td>
<td>Main text delimiters</td>
</tr>
<tr>
<td>&lt;BASEFONT FACE=&quot;Helvetica&quot; SIZE = 12&gt;</td>
<td>Body text font selection</td>
</tr>
<tr>
<td>&lt;FONT SIZE = +2&gt;... &lt;/FONT&gt;</td>
<td>Font type, size &amp; colour</td>
</tr>
<tr>
<td>&lt;Hx&gt;... &lt;/Hx&gt;</td>
<td>Subheading at level x</td>
</tr>
<tr>
<td>&lt;B&gt;... &lt;/B&gt;</td>
<td>Embolden font</td>
</tr>
<tr>
<td>&lt;I&gt;... &lt;/I&gt;</td>
<td>Italicize font</td>
</tr>
<tr>
<td>&lt;UL&gt;... &lt;/UL&gt;</td>
<td>Unordered list</td>
</tr>
<tr>
<td>&lt;OL&gt;... &lt;/OL&gt;</td>
<td>Ordered list</td>
</tr>
<tr>
<td>&lt;MENU&gt;... &lt;/MENU&gt;</td>
<td>Menu</td>
</tr>
<tr>
<td>&lt;LI&gt;</td>
<td>List start</td>
</tr>
<tr>
<td>&lt;BR&gt;</td>
<td>Break text, \n</td>
</tr>
<tr>
<td>&lt;P&gt;</td>
<td>New paragraph</td>
</tr>
<tr>
<td>&lt;HR&gt;</td>
<td>Horizontal line</td>
</tr>
<tr>
<td>&lt;PRE&gt;... &lt;/PRE&gt;</td>
<td>Nofill, preformatted</td>
</tr>
<tr>
<td>&lt;IMG SRC=&quot;...&quot;&gt;</td>
<td>Insert image file here</td>
</tr>
<tr>
<td>&lt;A HREF = &quot;http://www..&quot;&gt; [Press] &lt;/A&gt;</td>
<td>set up a Hyperlink</td>
</tr>
</tbody>
</table>

Starter set of HTML tags

CSA Rob Williams
Pearson Education (c) 2006
http proxy [URL] - The proxy command allows a proxy HTTP server to be defined which will be used in subsequent client commands. Providing a URL argument sets the proxy server. Setting the proxy to an empty string turns the proxy feature off.

http head url - The head command retrieves the HTTP header for the document located at URL.

http get url file - The get command retrieves the document located at URL. The body of the document is written to file. The command returns the HTTP header as described for the http head command above.

http post url filename_1 filename_2 - The post command posts the document in filename_1 to the location URL. The body of the returned document is written to filename_2. The command returns the HTTP header as described for the http head command above.

http put URL file - The put command copies the file into the URL. The command returns the HTTP header as described for the http head command above.

http delete URL - The delete command deletes the document at the URL. The command returns HTTP status information.

The %X variables are substituted before a script is evaluated:

%A - The network address of the client.

%P - The URL path requested by the requestor.

%S - The search path contained in the URL path.

Examples from Hypertext Transmission Protocol (http)
Attaching to a web server by telnet for an on-line session

Using http/GET in place of ftp/get
Client browser

DNS server

Multiple Google servers

Index servers each with 80 dual Pentium cards with 2 GB DRAM & 80 GB IDE disks in cooled racking

Schematic of Google search engine
Google data-flow activity diagram
## ISO Seven Layer OSI Model

<table>
<thead>
<tr>
<th>Layer</th>
<th>Sub-Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Application program</td>
</tr>
<tr>
<td>Presentation</td>
<td>transformation data formatting</td>
</tr>
<tr>
<td>Session</td>
<td>computer dialogue control</td>
</tr>
<tr>
<td>Transport</td>
<td>message &lt;=&gt; packets</td>
</tr>
<tr>
<td>Network</td>
<td>Virtual circuit routing control, packet sequencing</td>
</tr>
<tr>
<td>Link</td>
<td>Flow control, error/lost block detection</td>
</tr>
<tr>
<td>Physical</td>
<td>Voltage levels, plug pinouts</td>
</tr>
</tbody>
</table>

### Comparison of TCP/IP with ISO seven layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Sub-Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>TCP or UDP</td>
</tr>
<tr>
<td></td>
<td>IP</td>
</tr>
<tr>
<td></td>
<td>Network</td>
</tr>
</tbody>
</table>

CSA Rob Williams  
Pearson Education (c) 2006  
CSA ch 15 - p 215
16. CSA - Other Networks

POTS, the traditional telephone network

<table>
<thead>
<tr>
<th>1209</th>
<th>1336</th>
<th>1477 Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>#</td>
<td>0</td>
<td>941</td>
</tr>
</tbody>
</table>

DTMF, touch-tone signalling key pad

Digitization of telephone speech signals
Non-linear voice compression for transmission

\[ V_{out} = \frac{A \cdot V_{in}}{1 + \log A} \]
where \( A = 87.6 \)

2.048M
\[ \frac{64k}{64 \times 10^3} = \frac{10^3}{32} = 32 \text{ channels} \]

Multiplexor

30 channel TDM trunk line

Demultiplex

Time Division Multiplexing (TDM) for trunk line sharing
Space division circuit switching with control processor

Time Division Circuit Switching
Bands within the electromagnetic spectrum

Radio wave modulation using amplitude, frequency and phase techniques
Radio cell equipment and interconnection

<table>
<thead>
<tr>
<th>Frequency Range</th>
<th>Band</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1 - 0.3 GHz</td>
<td>VHF</td>
<td>Terrestrial television &amp; radio</td>
</tr>
<tr>
<td>0.3 - 1.0 GHz</td>
<td>UHF</td>
<td>Television, GSM mobile (0.9 GHz), packet radio, pagers</td>
</tr>
<tr>
<td>1.0 - 2.0 GHz</td>
<td></td>
<td>Navigation aids, GSM mobile (1.8 GHz)</td>
</tr>
<tr>
<td>2.4 - 2.5 GHz</td>
<td></td>
<td>Short range radio control (Bluetooth)</td>
</tr>
<tr>
<td>3.4 - 3.5 GHz</td>
<td></td>
<td>Neighbourhood antennae</td>
</tr>
</tbody>
</table>

Utilization of radio frequency bands
Cells with a repeat 7 pattern of radio frequencies

A cell arrangement with only repeat 4 pattern
gsm handset signal processing schematic

gsm handset functional modules
Packet structure for gsm voice transmissions

Circuit switched vs. packet switched message timing
The ATM frame structure

Schematic ATM router switch

Interconnection inside an ATM Banana Switch
ATM WAN linking together diverse LANs to form part of the Internet

Digital Paging word format
Application of packet radio data messaging

Narrow Band ISDN defined interfaces

2B-1D Narrowband ISDN protocol timing
Multi-level baseband encoding

<table>
<thead>
<tr>
<th>Binary data</th>
<th>3-level codes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>+0-</td>
</tr>
<tr>
<td>0001</td>
<td>-+0</td>
</tr>
<tr>
<td>0010</td>
<td>0+-</td>
</tr>
<tr>
<td>0011</td>
<td>+0-</td>
</tr>
<tr>
<td>0100</td>
<td>++0</td>
</tr>
<tr>
<td>0101</td>
<td>0++</td>
</tr>
<tr>
<td>0110</td>
<td>+0+</td>
</tr>
<tr>
<td>0111</td>
<td>+++</td>
</tr>
<tr>
<td>1000</td>
<td>++-</td>
</tr>
<tr>
<td>1001</td>
<td>-+-</td>
</tr>
<tr>
<td>1010</td>
<td>+-+</td>
</tr>
<tr>
<td>1011</td>
<td>+00</td>
</tr>
<tr>
<td>1100</td>
<td>0+0</td>
</tr>
<tr>
<td>1101</td>
<td>00+</td>
</tr>
<tr>
<td>1110</td>
<td>0+</td>
</tr>
<tr>
<td>1111</td>
<td>-0+</td>
</tr>
</tbody>
</table>

Three-level 4B3T encoding table
Digital Subscriber Line configuration

Subscriber line bandwidth allocation for ADSL
Cable TV and telephone distribution scheme

Bandwidth allocation for a cable TV network
CSA Ch 17

- Computers
  - Fetch-execute cycle
  - Hardware
    - CPU
      - Arithmetic Logic Unit
      - Control Unit
      - RISC features
      - ARM processor
      - Pentium
      - Itanium
    - Input-output
      - Parallel communication
      - Serial communication
    - Networking
      - Local Area Networks
        - Ethernet
        - USB
      - Wide Area Networks
        - Other Networks
        - Point to point
      - Visual output
    - Memory
      - Memory hierarchy
      - Cache and main memory
      - Disk filing
    - Parallel processing
  - Software
    - Operating systems
      - Unix
      - MS Windows
    - Tools
      - Compilers and assemblers
        - Subroutines and stacks
    - WIMPs
  - Users' viewpoints
    - Hardware engineer
    - HLL programmer
    - Systems administrator
    - Systems programmer
Types of computer operating systems
Unix boot sequence
Some Unix tools

<table>
<thead>
<tr>
<th>Tool</th>
<th>Function</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>awk</td>
<td>text processing language</td>
<td>cat file</td>
</tr>
<tr>
<td>cat</td>
<td>opens and concatenates files</td>
<td>cat header ch_01</td>
</tr>
<tr>
<td>diff</td>
<td>file comparison</td>
<td>diff ch_01.a ch_01.b</td>
</tr>
<tr>
<td>echo</td>
<td>repeats argument to stdout</td>
<td>echo $PATH</td>
</tr>
<tr>
<td>find</td>
<td>file search utility</td>
<td>find ” -name “<em>rob</em>” -print</td>
</tr>
<tr>
<td>grep</td>
<td>string (reg expr) search</td>
<td>grep ”rob” /etc/passwd</td>
</tr>
<tr>
<td>lpr</td>
<td>print demon</td>
<td>lpr -Pnts -#25 ~/Sheets/unix_intro</td>
</tr>
<tr>
<td>ls</td>
<td>directory listing</td>
<td>ls -al</td>
</tr>
<tr>
<td>more</td>
<td>text viewer</td>
<td>more book.txt</td>
</tr>
<tr>
<td>ps</td>
<td>process listing</td>
<td>ps -af</td>
</tr>
<tr>
<td>sed</td>
<td>stream editor</td>
<td>sed ’s/r-williams/rob.williams/g’ &lt;file1 &gt;file2</td>
</tr>
<tr>
<td>sort</td>
<td>string sorter</td>
<td>cat file</td>
</tr>
<tr>
<td>spell</td>
<td>spell checker</td>
<td>spell letter.txt</td>
</tr>
<tr>
<td>tr</td>
<td>transpose strings</td>
<td>tr -cs ‘A-Za-z’ ‘\012’</td>
</tr>
<tr>
<td>troff</td>
<td>text formatter</td>
<td>groff -petfH</td>
</tr>
<tr>
<td>uniq</td>
<td>repeated line detector</td>
<td>sort</td>
</tr>
<tr>
<td>users</td>
<td>network users list</td>
<td>users &gt; checkfile</td>
</tr>
<tr>
<td>wc</td>
<td>file sizer</td>
<td>wc -w assignment.txt</td>
</tr>
<tr>
<td>who</td>
<td>local user list</td>
<td>who</td>
</tr>
</tbody>
</table>

Typical operating system layered structure

- **API**
  - **GUI or Shell**
  - **API**
  - **GUI or Shell**

- **File manager**
- **Graphic primitives**
- **Scheduler**

- **Device drivers**
- **Memory allocation**
- **Task dispatcher**

**Computer hardware**
<table>
<thead>
<tr>
<th>PID - process id</th>
</tr>
</thead>
<tbody>
<tr>
<td>UID - owner</td>
</tr>
<tr>
<td>Process State</td>
</tr>
<tr>
<td>Semaphore ID</td>
</tr>
<tr>
<td>Signal ID</td>
</tr>
<tr>
<td>Memory needs</td>
</tr>
<tr>
<td>CODE SEG pointer</td>
</tr>
<tr>
<td>STACK SEG pointer</td>
</tr>
<tr>
<td>DATA SEG pointer</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Accounting info</td>
</tr>
<tr>
<td>File descriptors</td>
</tr>
<tr>
<td>Current directory</td>
</tr>
<tr>
<td>Task Queue pointers</td>
</tr>
</tbody>
</table>

**Summary contents of a TCB**
Tasks specified by their control blocks
Task Control Block queue
State Diagram showing the task lifecycle

Using `ps` & `wc` to count running tasks
<table>
<thead>
<tr>
<th><strong>UID</strong></th>
<th>(f,l)</th>
<th>The effective user ID number of the process (the login name is printed under the -f option).</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PID</strong></td>
<td>(all)</td>
<td>The process ID of the process (this number is used when killing a process).</td>
</tr>
<tr>
<td><strong>PPID</strong></td>
<td>(f,l)</td>
<td>The process ID of the parent process.</td>
</tr>
<tr>
<td><strong>STIME</strong></td>
<td>(f)</td>
<td>The starting time of the process, given in hours, minutes, and seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(A process begun more than 24 hours before the ps inquiry is displayed in days and months)</td>
</tr>
<tr>
<td><strong>TTY</strong></td>
<td>(all)</td>
<td>The controlling terminal for the process (“?” is shown when there is no controlling terminal as for background or daemon processes)</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>(all)</td>
<td>The cumulative execution time for the process.</td>
</tr>
<tr>
<td><strong>CMD</strong></td>
<td>(all)</td>
<td>The command name. (the full command name and its arguments, up to a limit of 80 chars, are printed with the -f option).</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td>(l)</td>
<td>The state of the process, (use the -f option):</td>
</tr>
<tr>
<td></td>
<td></td>
<td>O  Process is running on a processor.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S  Sleeping: process is waiting for an event to complete.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R  Runnable: process is on run queue.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Z  Zombie state: process terminated and parent is not waiting.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T  Process is stopped, either by a job control signal or because it is being traced.</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>(f,l)</td>
<td>Processor utilization for scheduling. Not printed when the -c option is used.</td>
</tr>
</tbody>
</table>

Information in the ps display, from the man page

- **Time-slicing**
- **Demand preemption**
- **Cooperative**
- **Interrupt driven**

Process scheduling techniques
### Displaying Unix task list using `ps`

<table>
<thead>
<tr>
<th>UID</th>
<th>PID</th>
<th>PPID</th>
<th>C</th>
<th>STIME</th>
<th>TTY</th>
<th>TIME</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:01</td>
<td>sched</td>
</tr>
<tr>
<td>root</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:02</td>
<td>/etc/init -</td>
</tr>
<tr>
<td>root</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>pageout</td>
</tr>
<tr>
<td>root</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>3:39</td>
<td>fsflush</td>
</tr>
<tr>
<td>root</td>
<td>322</td>
<td>297</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>145:37</td>
<td>/usr/openwin/bin/Xsun :0 -noban</td>
</tr>
<tr>
<td>root</td>
<td>122</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/usr/sbin/inetd -s</td>
</tr>
<tr>
<td>root</td>
<td>318</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/usr/lib/saf/sac -t 300</td>
</tr>
<tr>
<td>root</td>
<td>102</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/usr/sbin/rpcbind</td>
</tr>
<tr>
<td>root</td>
<td>112</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/usr/sbin/kerbd</td>
</tr>
<tr>
<td>root</td>
<td>110</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/usr/lib/netsvc/yp/ypbind</td>
</tr>
<tr>
<td>root</td>
<td>284</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:02</td>
<td>/sbin/vold</td>
</tr>
<tr>
<td>root</td>
<td>226</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/usr/lib/autoofs/automountd</td>
</tr>
<tr>
<td>root</td>
<td>240</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:01</td>
<td>/sbin/cron</td>
</tr>
<tr>
<td>root</td>
<td>230</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/sbin/syslogd</td>
</tr>
<tr>
<td>root</td>
<td>249</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:01</td>
<td>/sbin/nscd</td>
</tr>
<tr>
<td>root</td>
<td>259</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:01</td>
<td>/usr/lib/lpsched</td>
</tr>
<tr>
<td>root</td>
<td>3119</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>console</td>
<td>0:00</td>
<td>/usr/lib/saf/ttymon --g -h -p olves</td>
</tr>
<tr>
<td>root</td>
<td>274</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/usr/lib/utmpd</td>
</tr>
<tr>
<td>root</td>
<td>292</td>
<td>1</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/usr/lib/sendmail -q15m</td>
</tr>
<tr>
<td>root</td>
<td>321</td>
<td>318</td>
<td>0</td>
<td>Mar</td>
<td>?</td>
<td>0:00</td>
<td>/usr/lib/saf/ttymon</td>
</tr>
</tbody>
</table>

*rwilliam 340 323 0 Mar 16 ? 0:00 /bin/ksh /usr/dt/config/Xsession
rwilliam 408 407 0 Mar 16 ? 0:00 olwmslave
rwilliam 342 340 0 Mar 16 ? 0:00 /bin/ksh /usr/dt/config/Xsession
rwilliam 388 378 0 Mar 16 ? 0:00 /bin/ksh /usr/dt/config/Xsession
rwilliam 412 1 0 Mar 16 ?? 0:00 /usr/openwin/bin/cmdtool -Wp 0 0
rwilliam 378 342 0 Mar 16 ? 0:00 /bin/tcsh -c unsetenv _ PWD;
rwilliam 389 388 0 Mar 16 ? 0:00 /bin/ksh /home/staff/csm/csstaff/
rwilliam 407 389 0 Mar 16 ? 0:18 olwm -syncpid 406
rwilliam 19576 407 0 10:07:01 ?? 0:02 /usr/openwin/bin/xterm
rwilliam 415 412 0 Mar 16 pts/3 0:00 /bin/tcsh
rwilliam 19949 19577 0 14:04:16 pts/5 0:05 ghostview /tmp/tmp.ps
rwilliam 469 407 0 Mar 16 ? 10:35 /usr/local/bin/emacs
rwilliam 1061 1050 0 Mar 16 ? 0:00 (dns helper)
rwilliam 553 407 0 Mar 16 ? 0:21 /usr/openwin/bin/filemgr
rwilliam 11510 1 0 Mar 22 ? 4:08 /opt/simeon/bin/simeon.orig -u
rroot 20818 19577 1 18:01:46 pts/5 0:00 ps -Af
rwilliam 20304 19949 0 15:31:45 pts/5 0:06 gs -sDEVICE=x11 -DNOPAUSE -dQUIET
rwilliam 1050 407 0 Mar 16 ? 14:39 /usr/local/netscape/netscape
rwilliam 12251 1 0 Mar 22 ? 0:17 /usr/local/Acrobat4/Reader/spaces
rwilliam 19577 19576 0 10:07:02 pts/5 0:01 tcsh

---

CSA Rob Williams
Pearson Education (c) 2006

CSA ch 17 - p 240
sched: the O/S scheduler, notice the PID value, an important process, following boot
init: startup process from boot time, gets all the other Unix processes started
pageout: virtual memory page handler
fsflush: updates the super block and flushes data to disk
Xsun: X-window server
initt: Internet server daemon, provides remote services such as ftp, telnet, rlogin, talk
sac: port services access controller
rpcbind: address mapper for remote procedure calls
kerbd: source of kerberos unique keys, used for network user authentication
ypbind: NIS distributed password system
vold: file system (volume) management for CDROMs and floppy disk drives
automountd: daemon to handle remote file system mount/unmount requests
cron: schedules tasks to run at particular times
syslogd: system message handler and router
nscd: name service cache daemon
lpsched: printer service scheduler
ttymon: monitors terminal ports for activity
utmpd: user accounting daemon
sendmail: Internet mail server
ttymon: monitors terminal ports for activity
ksh: Korn shell user interface
fbconsole: console window
olmslave: Open Look X-window manager
ksh: second Korn shell user interface
ksh: third Korn shell user interface
cmdtool: command-tool window handler
tcsh: a tenex shell user interface
ksh: fourth Korn shell
olwm: Open Look Window Manager
xterm: X terminal window
tcsh: a tenex shell user interface
ghostview: PostScript screen viewer
emacs: the best editor!
dns: domain naming service for Internet name to IP number conversions
filemgr: drag ‘n drop file manager, useful for floppy disks
simeon: mail client
ps: this produced the process listing!
gs: ghostscript translator
netscape: Netscape Navigator Web browser
acroread: Adobe Acrobat reader for viewing pdf documents
tcsh: another user interface shell

Common Unix processes
Piping data between tasks in Unix

```
rob> world > hello.txt
rob> echo "Hello world!" > world.txt
rob> tr "\r" < ch_10.asc > ch_10
rob> cat header ch_* > book
rob> cat > letter << +++

? Dear Craig,
? Here is the book that I promised to send
? Rob
? +++
```

Redirecting data from tasks and files in Unix

Standard I/O for Unix processes

```
WAIT(sem_buff)
... //critical region code...
SIGNAL(sem_buff);
```

Semaphore operators, WAIT and SIGNAL
Semaphores protecting a cyclic data buffer

```c
#include <stdio.h>
#include <signal.h>
#define MYSIG 44

/* Signal handler function, evoked by sig 44
   reinstalls after each sig hit, prints number of hits */
void getsig(int s)
{
    static int count = 0;
    printf("signal %d again, %dth time \n", s, ++count);
    signal(MYSIG, getsig);
}

/* Process to demonstrate signals, sets up a sig handler to
   count the number of sig hits received. Loops forever.
   Start using "kbdcnt &" and make note of pid value returned
   Recommend to use "kill -44 pid" to send signals
   Remove using "kill -9 pid"
*/
int main(void)
{
    signal(MYSIG, getsig);
    printf("start counting kbd kills\n");
    while(1) {}
    return 0;
}
```

Unix signal handler to count and display signal hits
Using a signal to notify a Unix process

```plaintext
rob [262] gcc kbdcnt.c
rob [263] a.out &
rob [264] [1] 9539
rob [265] start counting kbd kills
kill -44 9539
rob [266] signal 44 again, 1th time
kill -44 9539
rob [267] signal 44 again, 2th time
kill -44 9539
rob [268] signal 44 again, 3th time
kill -44 9539
rob [269] signal 44 again, 4th time
```
#include <stdio.h>
#include <signal.h>
#include <errno.h>

#define PSIG 43 /* check the value of NSIG in */
    /* /usr/include/sys/signal.h */
#define CSIG 42 /* before choosing the signal values */

int ccount = 0;
int pcount = 0;
char str[] = "error message ";

void psigfunc(int s)
{
    pcount++;
    signal(CSIG, psigfunc);
}

void csigfunc(int s)
{
    ccount++;
    signal(PSIG, csigfunc);
}

main()
{
    int ke, tpid, ppid, cpid;

    ppid = getpid();
    cpid = fork(); /* spawn child process */
    if (cpid == -1)
    {
        printf("failed to fork
");  
        exit(1);
    }
if (cpid == 0 )
{
    /* Child process executes here */
    signal(PSIG, csigfunc);
    printf("Child started\n");
    while (1) {
        pause();
        printf("Child hit! count = %d\n", ccount);
        sleep(rand()%10);
        if( (kill(ppid, CSIG)) ) perror(str);
    }
}
else
{
    /* Parent process continues execution from here */
    signal(CSIG, psigfunc);
    printf("Parent started\n");
    while (1) {
        sleep(rand()%10);
        if( (kill(cpid, PSIG)) ) perror(str);
        pause();
        printf("Parent hit! count = %d\n", pcount);
    }
}

PSIG 43

Parent        Child

CSIG 42

Demonstrating the use of Signals by Unix Processes

Signal demonstrator program running on Unix

rob@olveston [52] a.out
Child started
Parent started
Child hit! count = 1
Parent hit! count = 1
Child hit! count = 2
Parent hit! count = 2
Child hit! count = 3
Parent hit! count = 3
Child hit! count = 4
Parent hit! count = 4
Child hit! count = 5
^C
rob@olveston [53]
Unix process creation using `fork()`

Virtual Address

Virtual to physical address translation

Interactive Unix Shells
Examples of shell command substitution

```
rob [52] grep "of the" 'ls | egrep "ch..." '
  ch_01: symbiosis established of the hardware and soft
  ch_11: as little as it is, the start of the applicatio
rob [53]
rob [53] mail 'cat mail_list' < message
rob [54]
rob[54] echo "There are 'who | wc -l' users logged in at 'date'"
  There are 12 users logged in at Tue June 20 20:23:55 BST 2000
rob [55]
```

echo "hello other window" > /dev/pts/0

value specifies which x-window to send to 0, 1, 2...
Unix device drivers in directory /dev
Unix termcap entry for an Xterm
<table>
<thead>
<tr>
<th>Code</th>
<th>Arg</th>
<th>Padding</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>al</td>
<td>str</td>
<td>(P*)</td>
<td>add new blank line</td>
</tr>
<tr>
<td>am</td>
<td>bool</td>
<td></td>
<td>terminal has automatic margins</td>
</tr>
<tr>
<td>bs</td>
<td>bool</td>
<td>(o)</td>
<td>terminal can backspace with <code>^H</code></td>
</tr>
<tr>
<td>cd</td>
<td>str</td>
<td>(P*)</td>
<td>clear to end of display</td>
</tr>
<tr>
<td>ce</td>
<td>str</td>
<td>(P)</td>
<td>clear to end of line</td>
</tr>
<tr>
<td>cl</td>
<td>str</td>
<td>(P*)</td>
<td>clear screen and home cursor</td>
</tr>
<tr>
<td>cm</td>
<td>str</td>
<td>(NP)</td>
<td>cursor move to row m, column n</td>
</tr>
<tr>
<td>cs</td>
<td>str</td>
<td>(NP)</td>
<td>change scroll region to lines m thro n</td>
</tr>
<tr>
<td>ct</td>
<td>str</td>
<td>(P)</td>
<td>clear all tab stops</td>
</tr>
<tr>
<td>dc</td>
<td>str</td>
<td>(P*)</td>
<td>delete character</td>
</tr>
<tr>
<td>dl</td>
<td>str</td>
<td>(P*)</td>
<td>delete line</td>
</tr>
<tr>
<td>im</td>
<td>str</td>
<td></td>
<td>enter insert mode</td>
</tr>
<tr>
<td>ho</td>
<td>str</td>
<td>(P)</td>
<td>home cursor</td>
</tr>
</tbody>
</table>

%d   decimal number starting at 0
%2   same as %2d
%3   same as %3d
%.   ASCII equiv
%+v  adds x then taken as %
%>xy if value is >x; then add y. no transmission
%r   reverse order of rows/columns
%i   origin is at 1,1 not 0,0
%%%  gives a single %
%n   XOR row and column (??)
%B   BCD format
%D   reverse coding

Some Unix termcap metacodes
18. CSA - Windows XP

Windows-NT/XP structure

Win32 and the Kernel
Displaying the PC task list using the Task Manager
Registry Editor windows displayed

HKEY_LOCAL_MACHINE holds the hardware configuration, installed device drivers, network protocols, software classes.
  Config configuration parameters for local computer
  Enum device configuration
  Hardwareserial port configuration
  Networkuser login information
  Securityremote administration permissions
  Softwareinstalled software
  Systembooting information

HKEY_CURRENT_CONFIG holds the current hardware configuration, where options exist

HKEY_CLASSES_ROOT holds document types, file associations, shell interface

HKEY_USERS holds login users' software preferences and desktop configuration

HKEY_CURRENT_USER holds copies of the preferences of the current user.

Registry top level keys
Setting network Shares and Permissions to a directory

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>User name</td>
<td>20 char</td>
</tr>
<tr>
<td>Passwords</td>
<td>14 char</td>
</tr>
<tr>
<td>Machine name</td>
<td>15 char</td>
</tr>
<tr>
<td>Workgroup names</td>
<td>15 char</td>
</tr>
<tr>
<td>Share names</td>
<td>12 char</td>
</tr>
</tbody>
</table>

Discrepancy in effective string lengths
Installing a shared directory as a local virtual drive
Computers

Fetch-execute cycle

Hardware

CPU

Arithmetic Logic Unit
Control Unit
RISC features
ARM processor
Pentium
Itanium

Input-output

Parallel communication
Serial communication
Networking

Local Area Networks
Ethernet
USB
Wide Area Networks
Other Networks
Point to point
Visual output

Memory

Memory hierarchy
Cache and main memory
Disk filing

Parallel processing

Software

Operating systems
Unix
MS Windows

Tools

Compilers and assemblers
Subroutines and stacks
WIMPs

Users' viewpoints

Hardware engineer
HLL programmer
Systems administrator
Systems programmer
## 19. CSA - Filing Systems

<table>
<thead>
<tr>
<th>Data organization</th>
<th>Application type</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>batch processing</td>
<td>Simple, efficient processing difficult</td>
<td>Maintenance needs sorting</td>
</tr>
<tr>
<td>Indexed-sequential</td>
<td>batch processing</td>
<td>Sequential and direct access no data sorting</td>
<td>Index takes up space less efficient</td>
</tr>
<tr>
<td>Direct</td>
<td>on-line</td>
<td>No data sorting fast access</td>
<td>Space inefficient inconvenient to use</td>
</tr>
<tr>
<td>Database</td>
<td>on-line</td>
<td>flexible access</td>
<td>Performance poor maintenance costs</td>
</tr>
</tbody>
</table>

### Data filing and databases

![Windows file browser showing the file hierarchy](image)

Windows file browser showing the file hierarchy
Sun File Manager display with floppy disk browser

Hard Disk with Four Partitions

Primary Partition
Secondary Partitions
Master Partition
Boot Record
Alternative layouts for blocks 1, 2, 3, 4, 5 & 7 to reduce access times (Head Movement Delays)
Program to check Partition Table and transfer to Boot Partition

Signature
AA55H

Partition Table
Partition 1
Partition 2
Partition 3
Partition 4

Code

Sector start Size Content
00H 1 Boot flag
01H 3 Start of partition
04H 1 System Flag
05H 3 End of Partition
08H 4 Start of Sector
OCH 4 # sectors

Head, Cylinder, Sector
H7 H6 H5 H4 H3 H2 H1 H0
C9 C8 S6 S5 S4 S3 S2 S1 S0
C7 C6 C5 C4 C3 C2 C1 C0

Boot Flag
- 00H Inactive (nonbootable),
  80H Active (bootable)

System Flag
- 01H FAT-12
  04H FAT-16
  05H Extended DOS partition
  08H AIX
  0AH OS/2
  0BH FAT-32
  DBH CP/M (!)
  83H Linux

Disk Master Partition Boot Record

- OS boot loader program
- # Hidden Sectors: 2 byte
- # Heads: 2 byte
- # Sectors / Track: 2 byte
- # sectors / FAT: 2 byte
- media byte (F8H): 1 byte
- # logical Sectors: 2 byte
- # Root DIR entries: 2 byte
- # FATS: 1 byte
- # Boot Sectors: 2 byte
- # Sectors / Cluster: 1 byte
- # Bytes / sector: 2 byte
- OEM Name/ID: 8 byte
- jmp to loader: 3 byte

A Partition Boot Record (non Master)

CSA Rob Williams
Pearson Education (c) 2006
Essential information for directory entries

Unix mount table

rob@milly [20] /usr/sbin/mount
/ on /dev/dsk/c0t0d0s0 read/write/setuid on Mon Jul 19 08:12:44 2000
/usr on /dev/dsk/c0t0d0s1 read/write/setuid on Mon Jul 19 08:12:44 2000
/proc on /proc read/write/setuid on Mon Jul 19 08:12:44 2000
/dev/fd on fd read/write/setuid on Mon Jul 19 08:12:44 2000
/var on /dev/dsk/c0t0d0s4 read/write/setuid on Mon Jul 19 08:12:44 2000
/cache/cache1 on /dev/dsk/c0t0d0s7 setuid/read/write on Mon Jul 19 08:13:4
/cache/cache2 on /dev/dsk/c0t0d0s7 setuid/read/write on Mon Jul 19 08:13:4
/cache/cache3 on /dev/dsk/c0t1d0s7 setuid/read/write on Mon Jul 19 08:13:4
/cache/cache4 on /dev/dsk/c0t1d0s7 setuid/read/write on Mon Jul 19 08:13:4
/cache/cache5 on /dev/dsk/c0t1d0s7 setuid/read/write on Mon Jul 19 08:13:4
/opt on /dev/dsk/c0t1d0s6 setuid/read/write on Mon Jul 19 08:13:45 2000
/tmp on /dev/dsk/c0t1d0s5 setuid/read/write on Mon Jul 19 08:13:45 2000
/tftpboot on /dev/dsk/c0t1d0s0 setuid/read/write on Mon Jul 19 08:13:45 19
/home/student/csm/BA/other on /dev/dsk/c0t1d0s4 nosuid/read/write/quot
/home/student/csm/BA/other on /dev/dsk/c0t1d0s5 nosuid/read/write/quot
/home/student/csm/PhD on /dev/dsk/c0t1d0s6 nosuid/read/write/quot
/home/student/csm/PhD on /dev/dsk/c0t1d0s3 nosuid/read/write/quot
...
FAT-16 Directory and File Allocation Table

<table>
<thead>
<tr>
<th>Sector Size</th>
<th>Sectors per Cluster</th>
<th>Cluster Size</th>
<th>Cluster Index Size</th>
<th>Nax Volume Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>512B</td>
<td>4</td>
<td>2kB</td>
<td>16bits</td>
<td>128MB</td>
</tr>
<tr>
<td>512B</td>
<td>16</td>
<td>8kB</td>
<td>16bits</td>
<td>512MB</td>
</tr>
<tr>
<td>512B</td>
<td>32</td>
<td>16kB</td>
<td>16bits</td>
<td>1GB</td>
</tr>
<tr>
<td>512B</td>
<td>64</td>
<td>32kB</td>
<td>16bits</td>
<td>2GB</td>
</tr>
<tr>
<td>512B</td>
<td>16</td>
<td>8kB</td>
<td>32bits</td>
<td>32TB</td>
</tr>
</tbody>
</table>

FAT cluster size and volume capacity
Unix file system inode structure

Unix inode file access records
Unix inode pointers indicating a file’s data blocks
Relating Unix directories to the inode blocks
Windows-NTFS Master File Table

File and directory access control options
RAID disk configurations
Setting File Access Permissions in Unix
20. CSA - Visual Output

<table>
<thead>
<tr>
<th>Display</th>
<th>Pixel data</th>
<th>Display size</th>
<th>Image size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full colour</td>
<td>24 bits</td>
<td>1024 x 768</td>
<td>2.25MB</td>
</tr>
<tr>
<td>Reduced-range colour</td>
<td>8 bits</td>
<td></td>
<td>0.75MB</td>
</tr>
<tr>
<td>Grey-scale monochrome</td>
<td>8 bits</td>
<td></td>
<td>0.75MB</td>
</tr>
<tr>
<td>Black-white monochrome</td>
<td>1 bit</td>
<td></td>
<td>96kB</td>
</tr>
</tbody>
</table>

Data requirements for different display types

Bitmapped raster display

Shadow mask inside a colour CRT
<table>
<thead>
<tr>
<th>Resolution</th>
<th>Vertical scan rate Hz</th>
<th>Horizontal scan rate kHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>640 x 480</td>
<td>60</td>
<td>31.5</td>
</tr>
<tr>
<td>640 x 480</td>
<td>72</td>
<td>37.8</td>
</tr>
<tr>
<td>800 x 600</td>
<td>75</td>
<td>46.9</td>
</tr>
<tr>
<td>800 x 600</td>
<td>85</td>
<td>53.7</td>
</tr>
<tr>
<td>1024 x 768</td>
<td>75</td>
<td>60.0</td>
</tr>
<tr>
<td>1024 x 768</td>
<td>85</td>
<td>68.8</td>
</tr>
<tr>
<td>1152 x 864</td>
<td>85</td>
<td>77.6</td>
</tr>
<tr>
<td>1280 x 1024</td>
<td>75</td>
<td>80.0</td>
</tr>
<tr>
<td>1280 x 1024</td>
<td>85</td>
<td>91.2</td>
</tr>
</tbody>
</table>

Normal CRT horizontal and vertical scan rates

The time period for writing a single pixel onto the screen can be estimated:

\[
\frac{1}{60 \times 1024 \times 768} = 21 \text{ ns/pixel}
\]
Twisted nematic LCD panels, showing the polarized, ribbed panels
Example pic script
An SVG script, rendered by Firefox
An SVGA graphics adapter card with AGP interface

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mono</td>
<td>1981</td>
</tr>
<tr>
<td>Hercules</td>
<td>1983</td>
</tr>
<tr>
<td>CGA</td>
<td>1983</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>EGA</td>
<td>1984</td>
</tr>
<tr>
<td>VGA</td>
<td>1987</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SVGA</td>
<td>1990</td>
</tr>
<tr>
<td></td>
<td>1995</td>
</tr>
<tr>
<td>XGA</td>
<td>1997</td>
</tr>
</tbody>
</table>

Evolving range of standards for screen display
### 29 pin DVI connector pinout and signal names

<table>
<thead>
<tr>
<th>Pin #</th>
<th>Signal Name</th>
<th>Pin #</th>
<th>Signal Name</th>
<th>Pin #</th>
<th>Signal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TMDS Data2-</td>
<td>9</td>
<td>TMDS Data1-</td>
<td>17</td>
<td>TMDS Data0-</td>
</tr>
<tr>
<td>2</td>
<td>TMDS Data2+</td>
<td>10</td>
<td>TMDS Data1+</td>
<td>18</td>
<td>TMDS Data0+</td>
</tr>
<tr>
<td>3</td>
<td>TMDS Data2/4 Shield</td>
<td>11</td>
<td>TMDS Data1/3 Shield</td>
<td>19</td>
<td>TMDS Data0/5 Shield</td>
</tr>
<tr>
<td>4</td>
<td>TMDS Data4-</td>
<td>12</td>
<td>TMDS Data3-</td>
<td>20</td>
<td>TMDS Data5-</td>
</tr>
<tr>
<td>5</td>
<td>TMDS Data4+</td>
<td>13</td>
<td>TMDS Data3+</td>
<td>21</td>
<td>TMDS Data5+</td>
</tr>
<tr>
<td>6</td>
<td>DDC Clock [SCL]</td>
<td>14</td>
<td>+5 V Power</td>
<td>22</td>
<td>TMDS Clock Shield</td>
</tr>
<tr>
<td>7</td>
<td>DDC Data [SDA]</td>
<td>15</td>
<td>Ground</td>
<td>23</td>
<td>TMDS Clock +</td>
</tr>
<tr>
<td>8</td>
<td>Analog Vert Sync</td>
<td>16</td>
<td>Hot Plug Detect</td>
<td>24</td>
<td>TMDS Clock -</td>
</tr>
<tr>
<td>C1</td>
<td>Analog Red</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>Analog Green</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C3</td>
<td>Analog Blue</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C4</td>
<td>Analog Horiz Sync</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C5</td>
<td>Analog GND Return</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 15 pin SVG connector pinout and signal names

<table>
<thead>
<tr>
<th>Pin</th>
<th>SVG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RED</td>
</tr>
<tr>
<td>2</td>
<td>GREEN</td>
</tr>
<tr>
<td>3</td>
<td>BLUE</td>
</tr>
<tr>
<td>4</td>
<td>ground</td>
</tr>
<tr>
<td>5</td>
<td>RED rtn</td>
</tr>
<tr>
<td>6</td>
<td>GREEN rtn</td>
</tr>
<tr>
<td>7</td>
<td>BLUE rtn</td>
</tr>
<tr>
<td>8</td>
<td>key-pin</td>
</tr>
<tr>
<td>9</td>
<td>SYNC rtn</td>
</tr>
<tr>
<td>10</td>
<td>Mon id</td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>H Sync</td>
</tr>
<tr>
<td>13</td>
<td>V Sync</td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

CSA Rob Williams

Pearson Education (c) 2006
Driving a colour screen

Driving a PC screen using a palette table
Synchronization of screen raster with display memory
Laser Printer

For a page represented as a 600 dpi image:

\[
\text{single A4 page image} = \frac{11 \times 7 \times 600 \times 600}{8} = 3.5 \text{Mbytes}
\]

The same page may be represented by far less data if it is ASCII coded:

Maximum number of characters on an A4 page = 60 \times 100 = 6000 char

The number of characters on a WP page is about 2500, 2.5 Kbytes of ASCII data.

Thus, the compressing ratio would be:

\[
\text{compression ratio} = \frac{2500}{3500000} = 0.0007
\]

Such a size reduction is certainly worth achieving but OCR software has only recently been improved enough to give acceptably fast and accurate performance.
PostScript development, script edited by emacs, rendered by ghostview
More PostScript, but now in colour!

gs -dNOPAUSE -dBATCH -r1200 -sDEVICE=pdfwrite -sOutputFile=ch_16.pdf ch_16.ps
Comparing file sizes: ASCII, ps and pdf
The "Hello World!" example in pdf code
Hello World!

Viewing the pdf File using Adobe Acrobat
Layout of a typical window scheme
<table>
<thead>
<tr>
<th>Virtual Device Interface (GDI)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bitmaps, Icons &amp; Metafiles</td>
</tr>
<tr>
<td>Creating Windows</td>
</tr>
<tr>
<td>Operating Windows</td>
</tr>
<tr>
<td>On-screen Menu handling</td>
</tr>
<tr>
<td>Dealing with Mouse and Keyboard Events</td>
</tr>
<tr>
<td>Handling Dialog Boxes</td>
</tr>
<tr>
<td>Timer Events</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Threads and Process Scheduling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception Messages</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Free Memory Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device handling</td>
</tr>
<tr>
<td>Printing and Text output</td>
</tr>
<tr>
<td>File Management</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data interchange through Clipboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE / DDE data interchange</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Parameter Registry Management</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Information</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DLL Management functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network Access Routines</td>
</tr>
<tr>
<td>Passing and processing Messages</td>
</tr>
<tr>
<td>Audio data management</td>
</tr>
</tbody>
</table>

**Win32 API facilities**
```c
#include <windows.h>

int WINAPI WinMain(HINSTANCE a, HINSTANCE b, LPSTR c, int d)
{
    MessageBox(NULL, "Hello World!", "WSM", MB_OK);
    return NULL;
}
```

Your first Windows application
X-Windows Programming

MMX and SSE data registers
21. CSA - RISC Processors

Functional hierarchy

Sun Microsystems’ UltraSparc II
1. Single length instruction codes
2. Single clock cycle execution period
3. Limited arithmetical complexity supported
4. Extensive supply of CPU registers
5. Limited repertoire of machine instructions
6. Only straightforward addressing modes supported
7. Hardware support for procedure handling
8. No structured data types recognised
9. Compiler supplied to support the architecture
10. Hardware CU, pipelined decoding
11. Simplified interrupt facilities

Principal features of RISC CPUs

Increasing clock speed of microprocessors
Stack passing Parameters are pushed up onto the stack before transferring control (jumping) to the subroutine. The subroutine code shares the same stack and can access the parameters through the stack or frame pointer. Copies of VALUE parameters are simply pushed onto the stack, while reference, VAR, parameters are 32 bit addresses pointing back at data. Stack frame setup overheads, and access times to non-local variables within scope are an issue.

Register passing Using CPU registers to hold the parameters is the fastest method, but limited by the number and size of registers available. Compilers select this technique only if a couple of simple (integer, char) variables are to be passed IN, and for the single OUT return value from functions.

Register windows This is a specialised stack technique used by SPARC processors to reduce the amount of stack PUSHing and POPping. By physically overlapping the stack frames for adjacent procedures, some of the local variables can be visible as parameters with no data copying. To further speed up the process, SPARC CPUs have fast stack caches.

Parameter blocks For machines without stacks, the problem of where to save the return address is solved by the CALL instruction inserting the return address at the top of the procedure code before transferring control. The parameters are cunningly inserted in a block immediately after the CALL instruction, thus giving the procedure access by using the return address as a pointer.

Global access Fortran and BASIC would rely on global data blocks visible to all code. This system of memory parameter blocks has been reinstituted for graphics and Windows programming, where the number of parameters is so great that little else could be suggested.

Variety of parameter passing methods
Multi-stage pipeline decoding - parallel processing

A 5 stage superscalar architecture with 3 execution units
<table>
<thead>
<tr>
<th>Dependency Type</th>
<th>Description</th>
<th>Example</th>
<th>Blocked until...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
<td>RAW, read after write</td>
<td>MOV EAX,10 ADD EBX,EAX</td>
<td>EAX is loaded</td>
</tr>
<tr>
<td></td>
<td>WAR, write after read</td>
<td>MOV EBX,EAX MOV EAX,10</td>
<td>EAX is read</td>
</tr>
<tr>
<td></td>
<td>WAW, write after write</td>
<td>MUL 100 ADD EAX,10</td>
<td>sequence correct</td>
</tr>
<tr>
<td><strong>Control</strong></td>
<td>the outcome of a previous instruction is essential for an instruction to complete</td>
<td>CMP AL,’q’ JZ exit</td>
<td>Z flag set</td>
</tr>
<tr>
<td><strong>Resource</strong></td>
<td>limited availability of hardware resources</td>
<td>floating-point arithmetic</td>
<td>unit free</td>
</tr>
</tbody>
</table>

**Instruction pipeline dependencies**
Use of Register File Store with superscalar processors
Mapping Table

<table>
<thead>
<tr>
<th>Valid Flag</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Register File

- r5
- r6
- r7
- r8
- r9
- r10
- r11
- r12
- r13
- r14

Register renaming

Control Unit Branch Prediction Table

<table>
<thead>
<tr>
<th>Conditional Instruction address</th>
<th>Branch Target address</th>
<th>Prediction Confidence Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td>Example core</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>v1</td>
<td>ARM1</td>
<td>Original processor, 26 bit address, coprocessor for Beeb</td>
</tr>
<tr>
<td>v2</td>
<td>ARM2</td>
<td>32 bit multiplier, coprocessor included Acorn Archmedies/A3000</td>
</tr>
<tr>
<td>v2a</td>
<td>ARM3</td>
<td>on-chip cache, semaphore support with swap instruction</td>
</tr>
<tr>
<td>v3</td>
<td>ARM6 &amp; ARM7DI</td>
<td>32 bit address, CPSR/SPSR, MMU first macro-cell product.</td>
</tr>
<tr>
<td>v3M</td>
<td>ARM3M</td>
<td>enhanced multiply with 64 bit result</td>
</tr>
<tr>
<td>v4</td>
<td>StrongArm</td>
<td>LD/ST for 8/16 bit values, system mode iPAQ PDA,</td>
</tr>
<tr>
<td>v4T</td>
<td>ARM7TDMI</td>
<td>Compressed Thumb instruction set, MULA used for many mobile handsets</td>
</tr>
<tr>
<td></td>
<td>ARM9TDMI</td>
<td></td>
</tr>
<tr>
<td>v5</td>
<td>XScale</td>
<td></td>
</tr>
<tr>
<td>v5TE</td>
<td>ARM9E &amp; ARM10E</td>
<td>better MUL, extra DSP instcns</td>
</tr>
<tr>
<td>v5TEJ</td>
<td>ARM7EJ &amp; ARM926EJ</td>
<td>Java option</td>
</tr>
<tr>
<td>v6</td>
<td>ARM11</td>
<td></td>
</tr>
</tbody>
</table>

**Historic revisions of ARM architectures**
ARM CPU registers, showing the alternate sets
Instruction formats for the ARM `ldr` & `str` instructions

Condition codes

<table>
<thead>
<tr>
<th>31</th>
<th>27</th>
<th>24</th>
<th>23</th>
<th>8</th>
<th>4</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Z</td>
<td>C</td>
<td>V</td>
<td>J</td>
<td>i</td>
<td>F</td>
</tr>
</tbody>
</table>

- Jazelle
- Saturated
- Overflow
- Carry out
- Zero value
- Negative value, msb=1

ARM program status register
<table>
<thead>
<tr>
<th>Opcode [31-28]</th>
<th>Mnm</th>
<th>Meaning</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>EQ</td>
<td>equal values</td>
<td>Z=1</td>
</tr>
<tr>
<td>0001</td>
<td>NE</td>
<td>not equal</td>
<td>Z=0</td>
</tr>
<tr>
<td>0010</td>
<td>CS</td>
<td>carry set</td>
<td>C=1</td>
</tr>
<tr>
<td>0011</td>
<td>CC</td>
<td>carry clear</td>
<td>C=0</td>
</tr>
<tr>
<td>0100</td>
<td>MI</td>
<td>negative value</td>
<td>N=1</td>
</tr>
<tr>
<td>0101</td>
<td>PL</td>
<td>positive value</td>
<td>N=0</td>
</tr>
<tr>
<td>0110</td>
<td>VS</td>
<td>overflow</td>
<td>V=1</td>
</tr>
<tr>
<td>0111</td>
<td>VC</td>
<td>no overflow</td>
<td>V=0</td>
</tr>
<tr>
<td>1000</td>
<td>HI</td>
<td>unsing higher</td>
<td>C=1 &amp;&amp; Z=0</td>
</tr>
<tr>
<td>1001</td>
<td>LS</td>
<td>unsing lower</td>
<td>C=0</td>
</tr>
<tr>
<td>1010</td>
<td>GE</td>
<td>greater or equal</td>
<td>N=V</td>
</tr>
<tr>
<td>1011</td>
<td>LT</td>
<td>less than</td>
<td>N!=V</td>
</tr>
<tr>
<td>1100</td>
<td>GT</td>
<td>greater than</td>
<td>Z=0 &amp;&amp; N=V</td>
</tr>
<tr>
<td>1101</td>
<td>LE</td>
<td>less or equal</td>
<td>Z=1</td>
</tr>
<tr>
<td>1110</td>
<td>AL</td>
<td>always</td>
<td></td>
</tr>
<tr>
<td>1111</td>
<td>NV</td>
<td>never use</td>
<td></td>
</tr>
</tbody>
</table>

**ARM condition codes**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov Rn, Rm</td>
<td>copy data between registers</td>
</tr>
<tr>
<td>ldr Rn, [Rm]</td>
<td>get a variable from memory</td>
</tr>
<tr>
<td>str Rn, [Rm]</td>
<td>put a variable back in memory</td>
</tr>
<tr>
<td>add R0, R1, R2</td>
<td>add two registers, result in third</td>
</tr>
<tr>
<td>cmp R0, R1</td>
<td>compare two registers</td>
</tr>
<tr>
<td>b addr</td>
<td>jump to relative location (+32 MB)</td>
</tr>
<tr>
<td>bl addr</td>
<td>call subroutine (+32MB)</td>
</tr>
<tr>
<td>mov R15, R14</td>
<td>return from subroutine</td>
</tr>
<tr>
<td>ldmf R13!, [Rm-Rn]</td>
<td>pop registers from stack</td>
</tr>
<tr>
<td>stmf R13!, [Rm-Rn]</td>
<td>push registers onto stack</td>
</tr>
<tr>
<td>ldr Rn, =constant</td>
<td></td>
</tr>
<tr>
<td>adr Rn, label</td>
<td></td>
</tr>
</tbody>
</table>

**Basic "starter" instructions for the ARM processor**
Instruction formats for move, arithmetic & logical instructions
The HP iPAQ hx4700 Pocket PC, with XScale PXA 270 processor

HP iPAQ hx2400 PCB. The 420 MHz XScale CPU is shrouded by emission reduction foil
Intel® PXA270 624MHz processor
64 MB Mobile SDRAM
128 MB Flash: 2 x 64 Mbyte (Intel RD48F4400L0zb0)
Up to 135 Mbyte of memory is available for user applications
100 mm transflective TFT VGA 64K color display (480x640)
Graphics controller (ATI Mobileon W3220)
Touch screen (Texas Instruments TSC2046 /SPI)
Touchpad (Synaptics NavPoint module /SPI),
Removable & rechargeable Lithium-Ion battery (1800 mAh)
Secure Digital (SDIO) slot
Compact Flash I & II (CF) slot
IrDA port (Exar XR16L580IL 16550-compatible)
WiFi LAN 802.11b capability (Texas Instruments TNETW1100B)
Bluetooth (Texas Instruments BRF6150USB)
RS232 port, 16550 compatible (PXA270)
Integrated microphone, speaker and stereo headset jack
Audio codec (AK4641)
USB (PXA270)
Weight: 186.7 g

HP iPAQ hx4700 series Pocket PC
<table>
<thead>
<tr>
<th>Bank</th>
<th>Sel line</th>
<th>MBytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>RAS/CAS3</td>
<td>128</td>
<td>DRAM bank 3</td>
</tr>
<tr>
<td>3</td>
<td>RAS/CAS2</td>
<td>128</td>
<td>DRAM bank 2</td>
</tr>
<tr>
<td>3</td>
<td>RAS/CAS1</td>
<td>128</td>
<td>DRAM bank 1</td>
</tr>
<tr>
<td>3</td>
<td>RAS/CAS0</td>
<td>128</td>
<td>DRAM bank 0</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>256</td>
<td>LCD &amp; DMA registers</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>256</td>
<td>Expansion &amp; memory</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>256</td>
<td>SCM registers</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>256</td>
<td>PM registers</td>
</tr>
<tr>
<td>1</td>
<td>CS5</td>
<td>128</td>
<td>Flash/SRAM bank 5</td>
</tr>
<tr>
<td>1</td>
<td>CS4</td>
<td>128</td>
<td>Flash/SRAM bank 4</td>
</tr>
<tr>
<td>0</td>
<td>PSTSEL</td>
<td>256</td>
<td>PCMIA socket 1</td>
</tr>
<tr>
<td>0</td>
<td>!PSTSEL</td>
<td>256</td>
<td>PCMIA socket 0</td>
</tr>
<tr>
<td>0</td>
<td>CS3</td>
<td>128</td>
<td>Flash/SRAM bank 3</td>
</tr>
<tr>
<td>0</td>
<td>CS2</td>
<td>128</td>
<td>Flash/SRAM bank 2</td>
</tr>
<tr>
<td>0</td>
<td>CS1</td>
<td>128</td>
<td>Flash/SRAM bank 1</td>
</tr>
<tr>
<td>0</td>
<td>CS0</td>
<td>128</td>
<td>Flash bank 0</td>
</tr>
</tbody>
</table>

**SA1110 StrongARM 4 GByte memory map**

**Physical Address**
FFFF_FFFFH

**internal to SA1110 StrongARM**

**0000_0000H**
Block diagram for the StrongARM core
Intel SA1110 StrongARM microcontroller
Operation of the Register File during procedure calls
CSA Ch 22

- CSA
  - Computers
    - Fetch-execute cycle
  - Hardware
    - CPU
      - Arithmetic Logic Unit
      - Control Unit
      - RISC features
      - ARM processor
      - Pentium
      - Itanium
    - Input-output
      - Parallel communication
      - Serial communication
      - Networking
        - Local Area Networks
          - Ethernet
          - USB
        - Wide Area Networks
        - Other Networks
          - Point to point
        - Visual output
    - Memory
      - Memory hierarchy
      - Cache and main memory
      - Disk filing
    - Parallel processing
  - Software
    - Operating systems
      - Unix
      - MS Windows
    - Tools
      - Compilers and assemblers
      - Subroutines and stacks
      - WIMP
    - Users' viewpoints
      - Hardware engineer
      - HLL programmer
      - Systems administrator
      - Systems programmer
## 22. CSA - The EPIC Itanium processor

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Icache</td>
<td>16 kbyte, 64 byte line</td>
</tr>
<tr>
<td>L1 Dcache</td>
<td>16 kbyte, 64 byte line, write-through</td>
</tr>
<tr>
<td>L2</td>
<td>256 kbyte, 128 byte line, write-back</td>
</tr>
<tr>
<td>L3</td>
<td>3-9 Mbyte, 128 byte line, write-back</td>
</tr>
<tr>
<td>Main memory</td>
<td>&lt;1 Pbyte ( (2^{64}) )</td>
</tr>
<tr>
<td>Clock</td>
<td>1.66 GHz</td>
</tr>
<tr>
<td>System bus</td>
<td>400 MHz, 128 bits wide</td>
</tr>
<tr>
<td>CPU power</td>
<td>100 watts</td>
</tr>
</tbody>
</table>

**Itanium 2 processor parameters**
Instruction Bundle for the IA-64 Architecture
Intel IA-64 / Itanium Register Set
<table>
<thead>
<tr>
<th>Register</th>
<th>name</th>
<th>title</th>
<th>class</th>
<th>usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gr(_0)</td>
<td>r0</td>
<td></td>
<td>constant</td>
<td>reads as zero</td>
</tr>
<tr>
<td>Gr(_1)</td>
<td>r1</td>
<td>gp</td>
<td>special</td>
<td>global data pointer</td>
</tr>
<tr>
<td>Gr(_2)-Gr(_3)</td>
<td>r2-r3</td>
<td>scratch</td>
<td></td>
<td>used with addl</td>
</tr>
<tr>
<td>Gr(_4)-Gr(_7)</td>
<td>r4-r7</td>
<td>preserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gr(<em>8)-Gr(</em>{11})</td>
<td>r8-r11</td>
<td>scratch</td>
<td></td>
<td>return values</td>
</tr>
<tr>
<td>Gr(_{12})</td>
<td>r12</td>
<td>sp</td>
<td>special</td>
<td>stack pointer</td>
</tr>
<tr>
<td>Gr(_{13})</td>
<td>r13</td>
<td>tp</td>
<td>special</td>
<td>thread pointer</td>
</tr>
<tr>
<td>Gr(<em>{14})-Gr(</em>{31})</td>
<td>r14-r31</td>
<td>scratch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gr(<em>{32})-Gr(</em>{39})</td>
<td>r32-r39</td>
<td>in0-in7</td>
<td>automatic</td>
<td>function params</td>
</tr>
<tr>
<td>Gr(<em>{32})-Gr(</em>{127})</td>
<td>r32-r127</td>
<td>loc0-loc95</td>
<td>automatic</td>
<td>input registers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>out0-out95</td>
<td>automatic</td>
<td>local registers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>output registers</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rotating registers (groups of 8)</td>
</tr>
</tbody>
</table>

**Asm programmers’ usage of Itanium registers**
<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ALU operations, arithmetic on integers and logic</td>
</tr>
<tr>
<td>I</td>
<td>multimedia, integer shifts, special register ops</td>
</tr>
<tr>
<td>M</td>
<td>memory load/store operations</td>
</tr>
<tr>
<td>B</td>
<td>branching, jumping and returning</td>
</tr>
<tr>
<td>F</td>
<td>floating-point operations</td>
</tr>
<tr>
<td>X</td>
<td>special instructions</td>
</tr>
</tbody>
</table>

Itanium instruction classes
<table>
<thead>
<tr>
<th>Template</th>
<th>Slot0</th>
<th>Slot1</th>
<th>Slot2</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>M-unit</td>
<td>I-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>01</td>
<td>M-unit</td>
<td>I-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>02</td>
<td>M-unit</td>
<td>I-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>03</td>
<td>M-unit</td>
<td>I-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>04</td>
<td>M-unit</td>
<td>L-unit</td>
<td>X-unit</td>
</tr>
<tr>
<td>05</td>
<td>M-unit</td>
<td>L-unit</td>
<td>X-unit</td>
</tr>
<tr>
<td>06</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>07</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>M-unit</td>
<td>M-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>09</td>
<td>M-unit</td>
<td>M-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>0A</td>
<td>M-unit</td>
<td>M-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>0B</td>
<td>M-unit</td>
<td>M-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>0C</td>
<td>M-unit</td>
<td>F-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>0D</td>
<td>M-unit</td>
<td>F-unit</td>
<td>I-unit</td>
</tr>
<tr>
<td>0E</td>
<td>M-unit</td>
<td>M-unit</td>
<td>F-unit</td>
</tr>
<tr>
<td>0F</td>
<td>M-unit</td>
<td>M-unit</td>
<td>F-unit</td>
</tr>
<tr>
<td>10</td>
<td>M-unit</td>
<td>I-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>11</td>
<td>M-unit</td>
<td>I-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>12</td>
<td>M-unit</td>
<td>B-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>13</td>
<td>M-unit</td>
<td>B-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>B-unit</td>
<td>B-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>17</td>
<td>B-unit</td>
<td>B-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>18</td>
<td>M-unit</td>
<td>M-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>19</td>
<td>M-unit</td>
<td>M-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>1A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1C</td>
<td>M-unit</td>
<td>F-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>1D</td>
<td>M-unit</td>
<td>F-unit</td>
<td>B-unit</td>
</tr>
<tr>
<td>1E</td>
<td>M-unit</td>
<td>I-unit</td>
<td>I-unit</td>
</tr>
</tbody>
</table>

Template field encoding for Itanium 2, showing the positions of inter-instruction stops
#include <stdio.h>

/* selection_sort.c  A good basic sort routine. 
Works by scanning up through the array finding the "smallest" item, which 
then swaps with the item at the start of the scan. It then scans again, 
starting at the second position, looking for the next smallest item... 
and so on. */

int selectionsort(char *pc[], int n) {
    int min, i, j, k;
    char *pctemp;
    for (i = 0; i < n; i++) {
        min = i;
        for (j = i + 1; j < n; j++)
            if (strcmp(pc[j], pc[min]) > 0) min = j;
        pctemp = pc[min];
        pc[min] = pc[i];
        pc[i] = pctemp;
        for (k = 0; k < n; k++)
            printf("%s ", pc[k]);
        printf("\n");
    }
    return 0;
}

void main() {
    int i;
    char *names[7], *testset[7] = {
        "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"
    }
    for (i = 0; i < 7; i++)
        names[i] = testset[i];
    printf("\n\nSelection Sort\n");
    i = selectionsort(names, 7);
}
.file "selection_sort.c"
.pred.safe_across_calls p1-p5,p16-p63
.section .rodata
.align 8
.LC0: stringz "%s 
.align 8
.LC1: stringz "\n"
.text
.align 16
.global selectionsort#
.proc selectionsort#

prologue 14, 34
.save ar.pfs, r35
.alloc r35 = ar.pfs, 2, 4, 2, 0
.vframe r36
.mov r36 = r12
.adds r12 = -48, r12
.mov r37 = r1
.save rp, r34
.mov r34 = b0
.body
;
.adds r14 = -32, r36
;
.st8 [r14] = r32
.adds r14 = -24, r36
;
.st4 [r14] = r33
.adds r14 = -16, r36
;
.st4 [r14] = r0

.prologue 14, 34
.save ar.pfs, r35
.alloc r35 = ar.pfs, 2, 4, 2, 0
.vframe r36
.mov r36 = r12
.adds r12 = -48, r12
.mov r37 = r1
.save rp, r34
.mov r34 = b0
.body
;
.adds r14 = -16, r36
;
.st8 [r14] = r32
.adds r14 = -24, r36
;
.st4 [r14] = r33
.adds r14 = -16, r36
;
.st4 [r14] = r0

FOR i loop

.adds r14 = -16, r36
.adds r15 = -24, r36
;;
.ld4 r16 = [r14]
.ld4 r14 = [r15]
;;
cmp4.gt p6, p7 = r14, r16
(p6) br.cond.dptk .L9
.br .L7
;;
Exit FOR i loop

.adds r15 = -20, r36
.adds r14 = -16, r36
;;
.ld4 r16 = [r14]
.ld4 r14 = [r15]
;;
.st4 [r15] = r14
.adds r15 = -12, r36
.adds r14 = -16, r36 ; get ipntr
;;
.ld4 r14 = [r14]
;;
.adds r14 = 1, r14 ; incr i
;;
.st4 [r15] = r14

.L6: ; FOR j loop

.adds r14 = -12, r36
.adds r15 = -24, r36
;;
.ld4 r16 = [r14] ; get npntr
.ld4 r14 = [r15] ; get jpntr
;;
cmp4.gt p6, p7 = r14, r16
(p6) br.cond.dptk .L9
.br .L7
;;
Exit j FOR loop

.adds r14 = -12, r36
;;
.ld4 r14 = [r14]
;;
sxt4 r14 = r14
;;
.shladd r15 = r14, 3, r0
.adds r16 = -32, r36
;;
.ld8 r14 = [r16]
;;
.add r16 = r15, r14
.adds r14 = -20, r36
;;
.ld4 r14 = [r14]
;;
sxt4 r14 = r14
;;
.shladd r15 = r14, 3, r0
.adds r17 = -32, r36
;;
.ld8 r14 = [r17]
;;
.add r14 = r15, r14
.ld8 r38 = [r16]
;;
.ld8 r39 = [r14]
.br.call.sptk.many b0 = strcmp#
.mov r1 = r37
.mov r14 = r8
;;
cmp4.ge p6, p7 = 0, r14
(p6) br.cond.dptk .L8
.adds r14 = -20, r36
.adds r15 = -12, r36
;;
.ld4 r15 = [r15]
;;
.st4 [r14] = r15
;;
.adds r14 = 1, r14 ; incr i
;;
.st4 [r15] = r14
.br .L6
;;
"CSA Rob Williams" "CSA ch 22 - p 318" "Pearson Education (c) 2006"
.L7: mov r16 = r36 ; get tempnptr
  adds r14 = -20, r36
  ;
  ld4 r14 = [r14]
  ;
  sx4 r14 = r14
  ;
  shladd r15 = r14, 3, r0
  adds r17 = -32, r36
  ;
  ld8 r14 = [r17]
  ;
  add r14 = r15, r14
  ;
  ld8 r14 = [r14]
  ;
  st8 [r16] = r14
  adds r14 = -20, r36
  ;
  ld4 r14 = [r14]
  ;
  sx4 r14 = r14
  ;
  shladd r15 = r14, 3, r0
  adds r17 = -32, r36
  ;
  ld8 r14 = [r17]
  ;
  add r15 = r15, r14
  ;
  ld14 = @ltoffx(.LC0), r1
  ;
  add r14 = 1, r14
  ;
  incr k
  ;
  st4 [r15] = r14
  ;
  br .L11
  ;
  .L11: ; FOR k loop
  adds r14 = -8, r36 ; get kpnter
  adds r15 = -24, r36
  ;
  ld4 r16 = [r14]
  ld4 r14 = [r15]
  ;
  cmp4.gt p6, p7 = r14, r16
  (p6) br.cond.dptk .L14
  br .L12 ; Exit k FOR loop
  ;
  .L14: adds r14 = -8, r36
  ;
  ld4 r14 = [r14]
  ;
  sx4 r14 = r14
  ;
  shladd r15 = r14, 3, r0
  adds r17 = -32, r36
  ;
  ld8 r14 = [r17]
  ;
  add r15 = r15, r14
  ;
  addl r14 = @ltoffx(.LC0), r1
  ;
  ld8.mov r38 = [r14], .LC0
  ld8 r39 = [r15]
  br.call.sptk.many b0 = printf#
  mov r1 = r37
  ;
  .L12: addl r14 = @ltoffx(.LC1), r1
  ;
  ld8.mov r38 = [r14], .LC1
  br.call.sptk.many b0 = printf#
  mov r1 = r37
  ;
  .L3: mov r14 = r0
  ;
  mov r8 = r14
  mov ar.pfs = r35
  ;
  save FP
  mov b0 = r34
  .restore sp
  mov r12 = r36
  br.ret.sptk.many b0
  ;
  .endp selectionsort#
.text
.globl main
.align 16
main:
    .prologue 14, 32
    .save ar.pfs, r33
    alloc r33 = ar.pfs, 0, 4, 2, 0
    .vframe r34
    mov r34 = r12 ; set FP from SP
    adds r12 = -144, r12 ; open stack frame
    mov r35 = r1
    .save rp, r32
    mov r32 = b0 ; save branch reg
    .body
    adds r15 = -48, r34
    mov r14 = @ltoffx(.LC2), r1 ; build tableptr
    addl r14 = @ltoffx(.LC3), r1 ; using offset+base ptr
    st8 [r14] = r14
    adds r16 = 8, r15 ; bump ptr
    addl r14 = @ltoffx(.LC3), r1
    ; "Tuesday"
    ld8.mov r14 = [r14], .LC3
    ;
    st8 [r16] = r14
    adds r16 = 16, r15
    addl r14 = @ltoffx(.LC4), r1
    ; "Wednesday"
    ld8.mov r14 = [r14], .LC4
    ;
    st8 [r16] = r14
    adds r16 = 24, r15
    addl r14 = @ltoffx(.LC5), r1
    ; "Thursday"
    ld8.mov r14 = [r14], .LC5
    ;
    st8 [r16] = r14
    adds r16 = 32, r15
    addl r14 = @ltoffx(.LC6), r1
    ; "Friday"
    ld8.mov r14 = [r14], .LC6
    ;
    st8 [r16] = r14
    adds r16 = 40, r15
    addl r14 = @ltoffx(.LC7), r1
    ; "Saturday"
    ld8.mov r14 = [r14], .LC7
    ;
    st8 [r16] = r14
    adds r16 = 48, r15
    addl r14 = @ltoffx(.LC8), r1
    ; "Sunday"
    ld8.mov r14 = [r14], .LC8
    ;
    st8 [r15] = r14
    adds r14 = -128, r34 ; build ipntr
    ;
    st4 [r14] = r0 ; clear i
    
    .L16: ; start of FOR i loop
        adds r15 = -128, r34
        ; names[]=testset[]
        ld4 r14 = [r15] ; get i
        ;
        cmp4.ge p6, p7 = 6, r14
        (p6) br.cond.dptk .L19 ; test end of FOR loop
        br .L17
    ;
    .L19: ; end of FOR loop
        adds r15 = -112, r34 ; get Destpntr
        adds r15 = -48, r34
        ; get Srcpntr
        ;
        ld4 r14 = [r16] ; get i
        ;
        sx7 r14 = r14 ; sign extend
        ;
        shladd r14 = r14, 3, r0
        ;
        add r16 = r14, r15
        ;
        adds r15 = -48, r34
        ;
        st4 [r16] = r14
        ;
        add r15 = 1, r14
        ;
        adds r16 = -128, r34
        ;
        br .L16 ; bottom of FOR
    ;
    .L17: ; start of FOR j loop
        adds r15 = -128, r34
        ; names[]=testset[]
        ld4 r14 = [r15] ; get i
        ;
        cmp4.ge p6, p7 = 6, r14
        (p6) br.cond.dptk .L19 ; test end of FOR loop
        br .L17
    ;
    .L19: ; end of FOR loop
        adds r15 = -112, r34 ; get Destpntr
        adds r15 = -48, r34
        ; get Srcpntr
        ;
        ld4 r14 = [r16] ; get i
        ;
        sx7 r14 = r14 ; sign extend
        ;
        shladd r14 = r14, 3, r0
        ;
        add r16 = r14, r15
        ;
        adds r15 = -48, r34
        ;
        st4 [r16] = r14
        ;
        add r15 = 1, r14
        ;
        adds r16 = -128, r34
        ;
        br .L16 ; bottom of FOR
    ;
    .endp main
    .ident "GCC: (GNU) 3.3.5 (Debian 1:3.3
.section .rodata ; Constants Section
.LC2: stringz "Monday"; constant strings store
.align 8
.LC3: stringz "Tuesday"
.align 8
.LC4: stringz "Wednesday"
.align 8
.LC5: stringz "Thursday"
.align 8
.LC6: stringz "Friday"
.align 8
.LC7: stringz "Saturday"
.align 8
.LC8: stringz "Sunday"
.align 8
.LC9: stringz "\n\nSelection Sort\n"

Itanium assembler code produced by gcc
Linux-64 with gvd/gdb debugger to watch Itanium code
CSA Ch 23

Computers
  Fetch-execute cycle
  Hardware
    CPU
      Arithmetic Logic Unit
      Control Unit
      RISC features
      ARM processor
      Pentium
      Itanium
  Input-output
    Parallel communication
    Serial communication
    Networking
      Local Area Networks
        Ethernet
        USB
      Wide Area Networks
      Other Networks
        Point to point
    Visual output
  Memory
    Memory hierarchy
    Cache and main memory
    Disk filing
  Parallel processing
  Software
    Operating systems
      Unix
      MS Windows
    Tools
      Compilers and assemblers
        Subroutines and stacks
      WIMPs
  Users' viewpoints
    Hardware engineer
    HLL programmer
    Systems administrator
    Systems programmer
Program has \( N \) instructions
Instructions take \( \tau \) secs each to complete
Uni-processor run time will be \( N \times \tau \)
IDEALLY, \( P \) processors could reduce run time to \( (N\tau)/P \)

But only a fraction of \textit{real} application code can be parallelized:

\[
total\_time = serial\_part + parallel\_part = N\tau f + \frac{N\tau(1 - f)}{P}
\]

While \( f \) is the fraction of the problem that must be carried out \textit{sequentially} due to data or control dependencies, \( (1 - f) \) is the fraction which can run in parallel. Note that \( f + (1 - f) \) evaluates to 1. \( P \) is the number of processors available.

The speed-up-ratio is then defined as the uni-processor-time divided by the (smaller!) multi-processor-time, which is:

\[
speed\_up\_ratio: S = \frac{N\tau}{N\tau f + \frac{N\tau(1 - f)}{P}} = \frac{1}{f + (1 - f)/P}
\]

Parallelizable code, Amdahl’s Law
Amdahl’s Law plotted for 1-15 processors and 0.1 - 0.9 compliant code
<table>
<thead>
<tr>
<th>Instruction stream</th>
<th>Data stream</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>SISD Single</td>
<td>Single</td>
<td>Personal PC</td>
</tr>
<tr>
<td>SIMD Single</td>
<td>Multiple</td>
<td>Vector processors</td>
</tr>
<tr>
<td>MISD Multiple</td>
<td>Single</td>
<td>Possibly none.</td>
</tr>
<tr>
<td>MIMD Multiple</td>
<td>Multiple</td>
<td>Central server</td>
</tr>
</tbody>
</table>

**Flynn’s processor taxonomy**

**Single bus, shared memory multi-processing (SMP)**

<table>
<thead>
<tr>
<th>Cache event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>Hit: cache read</td>
</tr>
<tr>
<td></td>
<td>Miss: main memory read</td>
</tr>
<tr>
<td></td>
<td>cache update</td>
</tr>
<tr>
<td>Write</td>
<td>Hit: main memory write</td>
</tr>
<tr>
<td></td>
<td>cache marked stale</td>
</tr>
<tr>
<td></td>
<td>main memory write</td>
</tr>
</tbody>
</table>

**Cache coherency protocol with write-through**
Hardware configuration for an MPI system

```c
int MPI_Send( void* sendBuf,  
int count,  
MPI_Datatype datatype,  
int destinationRank,  
int tag,  
MPI_Comm comm)  

int MPI_Recv( void* recvBuf,  
int count,  
MPI_Datatype datatype,  
int sourceRank,  
int tag,  
MPI_Comm comm,  
MPI_Status *status)  

int MPI_Bcast ( void* buffer,  
int count,  
MPI_Datatype datatype,  
int root,  
MPI_Comm comm )
```
The IBM/Sony Cell SPE unit
Cell schematic architecture

A single switched hub Cluster with Star topology
Grid computing