abs

compute integer absolute value

SYNOPSIS

#include <stdlib.h>

int abs(int i)

DESCRIPTION

The **abs** function returns the integer absolute value of an integer argument **i**.

RETURN VALUE

See above

RELATED FUNCTIONS

labs, fabs, fabsf

acos/acosf

compute arc-cosine

SYNOPSIS

```
#include <math.h>

double acos(double x)

float acosf(float x)
```

DESCRIPTION

The **acos** function returns the principal value of the arc-cosine of **x** (in radians). The **acosf** function does the same thing for a single precision argument.

RETURN VALUE

A domain error ("errno" is set to EDOM), occurs if the absolute value of **x** is greater than one. Otherwise, the return value falls in the range [0,PI] radians.

RELATED FUNCTIONS

asin, atan, cos, sin, tan

addfree

add memory for heap allocation

SYNOPSIS

```
#include <stdlib.h>

void addfree(void *ptr, size_t size)
```

DESCRIPTION

Add a region of memory to be used by the "C" heap allocation routines. **ptr** should point to the start of the region to add, **size** is the length in bytes. Regions added to the heap in this fashion are disjoint from other parts of the heap. This means blocks allocated from this heap region aren't mergable with those from other regions when freed (for the purpose of creating a block larger than any one region).

You should ensure that the region of memory you are adding isn't used in any other way by your program, including as part of the default heap!

RETURN VALUE

None

RELATED FUNCTIONS

calloc, cfree, free, malloc, realloc

asctime

format broken-down time

SYNOPSIS

#include <time.h>

char *asctime(const struct tm *timeptr)

DESCRIPTION

The **asctime** function converts the broken-down time in the structure pointed to by **timeptr**, into an ASCII string representation. A sample of the string representation:

"Sat Aug 15 01:03:51 1972\n\0"

RETURN VALUE

The **asctime** function returns a pointer to the converted string. Note that the string is stored in a static data area which is shared with the "ctime" function. The string will be overwritten upon subsequent calls to either function.

RELATED FUNCTIONS

ctime, gmtime, localtime, time

asin/asinf

compute arc-sine

SYNOPSIS

#include <math.h>

double asin(double x)

float asinf(float x)

DESCRIPTION

The **asin** function returns the principal value of the arc-sine of x (in radians). The **asinf** function does the same thing for a single precision argument.

RETURN VALUE

A domain error ("errno" is set to EDOM), occurs if the absolute value of **x** is greater than (1). Otherwise, the return value falls in the range [-PI/2,+PI/2] radians.

RELATED FUNCTIONS

acos, atan, cos, sin, tan

atan/atanf

compute arc-tangent

SYNOPSIS

#include <math.h>

double atan(double x)

float atanf(float x)

DESCRIPTION

The **atan** function returns the principal value of the arc-tangent of **x** in radians. The **atanf** function does the same thing for a single precision argument.

RETURN VALUE

The return value falls in the range [-PI/2,+PI/2] radians.

RELATED FUNCTIONS

acos, asin, atan2, cos, sin, tan

atan2/atan2f

compute arc-tangent of x/y

SYNOPSIS

```
#include <math.h>

double atan2(double x, double y)

float atan2f(float x, float y)
```

DESCRIPTION

The **atan2** function returns the principal value of the arc-tangent of **x/y** using the signs of both arguments to determine the quadrant of the return value (in radians). The **atan2f** function does the same thing for single precision arguments.

RETURN VALUE

A domain error ("errno" is set to EDOM), occurs if both **x** and **y** are zero. Otherwise, the return value falls in the range [-PI,+PI] radians.

RELATED FUNCTIONS

atan

atof/atoi/atol

convert strings to numbers

SYNOPSIS

```
#include <stdlib.h>

double atof(const char *cptr)

int atoi(const char *cptr)

long atol(const char *cptr)
```

DESCRIPTION

The **atof** function converts an ASCII numeric string pointed to by **cptr** into the equivalent floating point number. The **atoi** and **atol** functions do the same thing for integers, and long integers, respectively. In all cases, the strings are assumed to be base 10, whitespace is allowed before the numeric string, and the first unrecognized character stops the conversion.

atof recognizes a numeric string consisting of an optional sign, a digit sequence with optional decimal point, and an optional exponent ('e' or 'E', optional sign and exponent digit sequence).

atoi and **atol** recognize integral numbers consisting of an optional sign and digit sequence.

RETURN VALUE

The converted value.

RELATED FUNCTIONS

strtod, strtol, strtoul

bcmp

compare bytes

SYNOPSIS

```
#include <string.h>

int bcmp(const void *s1, const void *s2, size_t n)
```

DESCRIPTION

The **bcmp** function compares byte string **s1** against byte string **s2** for **n** bytes, returning zero if they are identical, nonzero otherwise.

RETURN VALUE

See above

RELATED FUNCTIONS

memcmp

bcopy

string copy with length

SYNOPSIS

#include <string.h>

void *bcopy(const void *src, void *dst, size_t n)

DESCRIPTION

The **bcopy** function copies **n** bytes from the location pointed to by **src** to the location pointed to by **dst**.

The **bcopy** function is equivalent to "memcpy", but with a reversed **src** and **dst** argument order.

This function is implemented "inline". To get a functional version, precede your call with:

#undef bcopy

RETURN VALUE

Returns the value of **dst**.

RELATED FUNCTIONS

memcpy, memmove, strcpy

BitCnt

count the number of 1 bits set

SYNOPSIS

```
#include <conc.h>

int BitCnt(int i)
```

DESCRIPTION

Supplied only for Transputers which support the corresponding instruction. The **BitCnt** function takes a single integer argument and returns the number of 1 bits set.

This function is implemented "inline". To get a functional version, precede your call with:

```
#undef BitCnt
```

RETURN VALUE

See above

RELATED FUNCTIONS

BitRevNBits, BitRevWord

BitRevNBits

variable bit reversal

SYNOPSIS

```
#include <conc.h>

int BitRevNBits(int numbits, int data)
```

DESCRIPTION

Supplied only for Transputers which support the corresponding instruction. The **BitRevNBits** function takes two integer arguments; **numbits** specifies how many of the bits in **data** should be reversed (counting from least significant bit). The result of the reversal is returned. All high order bits not involved in the reversal are zeroed.

This function is implemented "inline". To get a functional version, precede your call with:

```
#undef BitRevNBits
```

RETURN VALUE

See above

RELATED FUNCTIONS

BitCnt, BitRevWord

BitRevWord

reverse bits in word

SYNOPSIS

```
#include <conc.h>

int BitRevWord(int data)
```

DESCRIPTION

Supplied only for Transputers which support the corresponding instruction. The **BitRevWord** function reverses the bit pattern in **data** end-for-end, and returns the result.

This function is implemented "inline". To get a functional version, precede your call with:

```
#undef BitRevWord
```

RETURN VALUE

See above

RELATED FUNCTIONS

BitCnt, BitRevNBits

bzero

set a range of memory to a value

SYNOPSIS

```
#include <string.h>

void bzero(void *dst, size_t n)
```

DESCRIPTION

The **bzero** function writes **n** zero bytes into memory starting at **dst**.

RETURN VALUE

None

RELATED FUNCTIONS

memset

cabs/cabsf

compute complex absolute value

SYNOPSIS

```
#include <math.h>

double cabs(COMPLEX x)

float cabsf(COMPLEXF x)
```

DESCRIPTION

The **cabs** function returns the absolute value of a complex number **x**. A call to **cabs** is equivalent to the following expression:

```
hypot(x.real,x.imag)
```

The **cabsf** function does the same thing for a single precision argument.

RETURN VALUE

See above

RELATED FUNCTIONS

fft, ifft

calloc

allocate and initialize heap memory

SYNOPSIS

```
#include <stdlib.h>

void *calloc(size_t nmemb, size_t size)
```

DESCRIPTION

The **calloc** function allocates, and bitwise zeroes, a region of heap memory large enough to hold **nmemb** items of **size** bytes each.

RETURN VALUE

If it is impossible to satisfy the request, or **size** is zero, a NULL pointer is returned. Otherwise, a pointer is returned to the start of the allocated region.

RELATED FUNCTIONS

addfree, cfree, free, malloc, realloc

ceil/ceilf

compute "ceiling"

SYNOPSIS

```
#include <math.h>

double ceil(double x)

float ceilf(float x)
```

DESCRIPTION

The **ceil** and **ceilf** functions return the smallest floating point integral value not less than **x**.

RETURN VALUE

See above

RELATED FUNCTIONS

floor, floorf

cfree

free heap memory

SYNOPSIS

#include <stdlib.h>

void cfree(void *ptr)

DESCRIPTION

The **cfree** function returns the previously allocated region of heap memory pointed to by **ptr** to the heap free storage pool.

The value of **ptr** must be the result of an earlier call to "calloc", "malloc", or "realloc"; or the results are undefined.

The **cfree** function is equivalent to the "free" function.

RETURN VALUE

None

RELATED FUNCTIONS

addfree, calloc, free, malloc, realloc

ChanAlloc/ChanFree

communication channel allocation

SYNOPSIS

```
#include <conc.h>

Channel *ChanAlloc(void)

void ChanFree(Channel *ptr)
```

DESCRIPTION

The **ChanAlloc** function returns a pointer to an initialized software "channel". The memory for the "channel" is obtained from the heap.

The **ChanFree** function takes a "channel" previously allocated using **ChanAlloc** and returns it to the heap free storage pool.

Note that "channels" may also be statically allocated (either globally or as "auto" variables), if the dynamic creation and destruction allowed by **ChanAlloc** and **ChanFree** isn't required.

For additional information see the included paper by Jeff Mock: **Processes, Channels, and Semaphores**.

RETURN VALUE

ChanAlloc either returns a pointer to the allocated "channel", or NULL if no heap memory was available.

There is no return value from **ChanFree**.

RELATED FUNCTIONS

ChanIn, ChanInChanFail, ChanInChar, ChanInInt, ChanInTimeFail, ChanOut, ChanOutChanFail, ChanOutChar, ChanOutInt, ChanOutTimeFail, ChanReset

ChanIn/ChanInChar/ChanInInt

reading from channels

SYNOPSIS

```
#include <conc.h>

void ChanIn(Channel *c, void *ptr, int n)

char ChanInChar(Channel *c)

int ChanInInt(Channel *c)
```

DESCRIPTION

The **ChanIn** function reads **n** bytes of data, from the "channel" pointed to by **c**, to the buffer pointed to by **ptr**. The **ChanInChar** and **ChanInInt** functions may be used to read, and return, the value of a byte or word, respectively, read from the "channel" pointed to by **c**.

The Transputer supports a message passing paradigm, in hardware, using the concept of "channels". Briefly, "channels" are unidirectional message passing funnels between two processes, or processors (consult the appropriate INMOS documentation for a more detailed explanation). The **ChanIn**, **ChanInChar**, and **ChanInInt** functions allow for the reading of data from a "channel"; the complementary "ChanOut", "ChanOutChar" and "ChanOutInt" routines may be used to write data to a "channel".

In the Transputer model, "channel" communication may also be used as a scheduling mechanism. This works because no buffering is associated with a "channel"; the communication occurs only when both the reading and writing processes are ready.

In Transputer terminology, "channels" are classified as either "hard" or "soft". "hard channels" are those associated with the physical links between Transputers, "soft channels" are those used for intra-processor communication. One implementation difference between these two classes of channels may sometimes be usefully exploited:

The message length for the reading and writing process doing "channel" communication must be identical for "soft channels", but may be different if "hard channels" are used. For example, this means that a process using a "hard channel" can send a 6 byte message as one "ChanOut" call, while the corresponding **ChanIn** call on the connected processor could be broken into two **ChanIn** calls for 3 bytes each. This feature is particularly useful when implementing packet based communication protocols (the overhead portion of the packet can be received separately, and used to configure the length of the **ChanIn** call used to read the rest of the packet).

In order to be used, "channels" must be initialized. This is accomplished automatically by the "ChanAlloc" function, and should also be done by the programmer for "channels" created in other ways. For example, a "channel" might be declared and initialized in "C" using the following statement:

```
Channel xyz = NOPROCESS;
```

Where "xyz" is the name of the "channel", and the macro NOPROCESS is used to supply the correct initial value. The NOPROCESS macro definition, and symbolic definitions for the addresses of the "hard channels", may be found in the "conc.h" include file.

These functions are implemented "inline". To get a functional version, precede your call with:

```
#undef ChanIn
```

Or

```
#undef ChanInChar
```

Or

```
#undef ChanInInt
```

For additional information about the "channel" primitives, see the included paper by Jeff Mock: **Processes, Channels, and Semaphores**.

RETURN VALUE

See above

RELATED FUNCTIONS

ChanAlloc, ChanFree, ChanInChanFail, ChanInTimeFail, ChanOut, ChanOutChanFail, ChanOutChar, ChanOutInt, ChanOutTimeFail, ChanReset

ChanInChanFail/ChanInTimeFail

error read from channel

SYNOPSIS

```
#include <conc.h>

int ChanInChanFail(Channel *c1, void *ptr, int n, Channel *c2)

int ChanInTimeFail(Channel *c, void *ptr, int n, int t)
```

DESCRIPTION

The **ChanInChanFail** function is equivalent to the "ChanIn" function up through the first three parameters. In addition, a fourth parameter has been added (**c2**), which allows the "channel" read operation on **c1** to be aborted when an "int" is written to the auxiliary channel **c2** (and thus **c2** becomes ready for reading by **ChanInChanFail**). This allows another process to "timeout" a communication operation. The **ChanInChanFail** function automatically resets channel **c1** if the communication is aborted. Note that if channel **c1** is a "hard" channel, and communication was in progress, the reset will cause the communication operation to never complete (from the point of view of the other processor).

The **ChanInTimeFail** function is like **ChanInChanFail**, except the fourth parameter is a "time" value which is compared to the hardware clock associated with the current priority level. When the hardware clock is "after" the specified value **t**, the communication is aborted and the "channel" is reset.

For additional information about the "channel" primitives, see the included paper by Jeff Mock: **Processes, Channels, and Semaphores**.

RETURN VALUE

Either function returns zero if the communication completed, or one if the communication was aborted.

RELATED FUNCTIONS

ChanAlloc, ChanFree, ChanIn, ChanInChar, ChanInInt, ChanOut, ChanOutChanFail, ChanOutChar, ChanOutInt, ChanOutTimeFail, ChanReset

ChanOut/ChanOutChar/ChanOutInt
writing to channels

SYNOPSIS

```
#include <conc.h>

void ChanOut(Channel *c, void *ptr, int n)

void ChanOutChar(Channel *c, int byte)

void ChanOutInt(Channel *c, int word)
```

DESCRIPTION

The **ChanOut** function writes **n** bytes of data, to the "channel" pointed to by **c**, from the buffer pointed to by **ptr**. The **ChanOutChar** and **ChanOutInt** functions may be used to write the value of a **byte** or **word**, respectively, to the "channel" pointed to by **c**.

See the discussion for the corresponding read operations ("ChanIn" ...), for a more lengthy introduction.

These functions are implemented "inline". To get a functional version, precede your call with:

```
#undef ChanOut
```

Or

```
#undef ChanOutChar
```

Or

```
#undef ChanOutInt
```

For additional information about the "channel" primitives, see the included paper by Jeff Mock: **Processes, Channels, and Semaphores**.

RETURN VALUE

None

RELATED FUNCTIONS

ChanAlloc, ChanFree, ChanIn, ChanInChanFail, ChanInChar, ChanInInt, ChanInTimeFail, ChanOutChanFail, ChanOutTimeFail, ChanReset

ChanOutChanFail/ChanOutTimeFail
error write to channel

SYNOPSIS

```
#include <conc.h>

int ChanOutChanFail(Channel *c1, void *ptr, int n, Channel *c2)

int ChanOutTimeFail(Channel *c, void *ptr, int n, int t)
```

DESCRIPTION

The **ChanOutChanFail** function is equivalent to the "ChanOut" function up through the first three parameters. In addition, a fourth parameter has been added (**c2**), which allows the "channel" write operation on **c1** to be aborted when an "int" is written to the auxiliary channel **c2** (and thus **c2** becomes ready for reading by **ChanOutChanFail**). This allows another process to "timeout" a communication operation. The **ChanOutChanFail** function automatically resets channel **c1** if the communication is aborted. Note that if channel **c1** is a "hard" channel, and communication was in progress, the reset will cause the communication operation to never complete (from the point of view of the other processor).

The **ChanOutTimeFail** function is like **ChanOutChanFail**, except the fourth parameter is a "time" value which is compared to the hardware clock associated with the current priority level. When the hardware clock is "after" the specified value **t**, the communication is aborted and the "channel" is reset.

For additional information about the "channel" primitives, see the included paper by Jeff Mock: **Processes, Channels, and Semaphores**.

RETURN VALUE

Either function returns zero if the communication completed, or one if the communication was aborted.

RELATED FUNCTIONS

ChanAlloc, ChanFree, ChanIn, ChanInChanFail, ChanInChar, ChanInInt, ChanInTimeFail, ChanOut, ChanOutChar, ChanOutInt, ChanReset

ChanReset

reset a channel

SYNOPSIS

```
#include <conc.h>

int ChanReset(Channel *c)
```

DESCRIPTION

The **ChanReset** function is an extremely low level function which is used to reset and initialize a Transputer channel. The value returned reflects the current status of the channel.

Under normal circumstances, use of this function should NEVER be required, and it is included only to provide a "C" binding for the Transputer "resetch" instruction. Consult the appropriate INMOS instruction set documentation for further information.

This function is implemented "inline". To get a functional version, precede your call with:

```
#undef ChanReset
```

RETURN VALUE

The current status of the channel being reset. This is either the process descriptor of the process blocked on the channel, or the value of the NOPROCESS macro (if the channel was not in use).

RELATED FUNCTIONS

ChanAlloc, ChanFree, ChanIn, ChanInChanFail, ChanInChar, ChanInInt, ChanInTimeFail, ChanOut, ChanOutChanFail, ChanOutChar, ChanOutInt, ChanOutTimeFail

_cioext

host specific server extensions

SYNOPSIS

```
#include <cioext.h>

int _cioext(int fn, void *ibuf, int ilen, void *obuf, int olen)
```

DESCRIPTION

The **_cioext** function provides a user extensible, remote procedure call mechanism for use with the CIO host I/O server. The arguments to **_cioext** are:

1. **fn** - The desired user extension function code.

2. **ibuf** - A pointer to the input buffer for the user extension function.

3. **ilen** - The number of bytes in **ibuf**. The input buffer must be shorter than MAX_MSG_DATA as defined in cio.h (25000 bytes in the standard **Transputer Toolset** release).

4. **obuf** - A pointer to the output buffer for the user extension function.

4. **olen** - The number of bytes to return to **obuf**. The output buffer must be shorter than MAX_MSG_DATA as defined in cio.h (25000 bytes in the standard **Transputer Toolset** release).

The actual functions which are available are dependent on the capabilities implemented in the "cioext.c" file (which is linked with CIO). See the **CIO "C" I/O Driver User Guide** manual for further information.

RETURN VALUE

If the input or output buffers are too large -1 is returned. Otherwise the return value is completely dependent on the implementation of the specific function being invoked.

RELATED FUNCTIONS

None

clearerr

clear stream status

SYNOPSIS

```
#include <stdio.h>

void clearerr(FILE *stream)
```

DESCRIPTION

The **clearerr** function resets the "eof" and "error" status flags associated with the specified **stream**.

RETURN VALUE

None

RELATED FUNCTIONS

feof, ferror, rewind

close

low level file close

SYNOPSIS

#include <stdio.h>

int close(int handle)

DESCRIPTION

Closes the file with file descriptor **handle**.

The file must have been previously opened by a call to "open" or "creat".

Note that termination of a program via "exit", or a return from "main", also closes all open files.

As this function is not supported in the ANSI standard library, its use in new work is discouraged (see the "fopen", "fclose", "fread", and "fwrite" functions for the recommended replacements).

RETURN VALUE

The **close** function returns zero if the file was successfully closed. A return value of (-1), indicates an error, and "errno" is set appropriately.

RELATED FUNCTIONS

creat, dup, dup2, open, read, write

cos/cosf

compute cosine

SYNOPSIS

#include <math.h>

double cos(double x)

float cosf(float x)

DESCRIPTION

The **cos** function returns the cosine of **x** (measured in radians). The **cosf** function does the same thing for a single precision argument.

RETURN VALUE

A large magnitude argument may yield a result with little significance.

RELATED FUNCTIONS

acos, asin, atan, sin, tan

cosh/coshf

compute hyperbolic cosine

SYNOPSIS

#include <math.h>

double cosh(double x)

float coshf(float x)

DESCRIPTION

The **cosh** function returns the hyperbolic cosine of **x**. The **coshf** function does the same thing for a single precision argument.

RETURN VALUE

A range error ("errno" is set to ERANGE), occurs if the magnitude of **x** is too large.

RELATED FUNCTIONS

sinh, tanh

creat

low level file creation

SYNOPSIS

```
#include <stdio.h>
#include <fcntl.h>

int creat(char *path, int mode)
```

DESCRIPTION

The **creat** function either creates a new file, or opens and truncates an existing file. If the file specified by **path** doesn't already exist, it is created with read/write permission as set by **mode**. If the file already exists, and its permissions allow writing, the file is truncated to zero length and opened for writing.

The **mode** parameter applies only to newly created files, and determines what permission setting the file should be given when closed.

The following symbolic bit mask macro definitions are used with **creat**:

S_IREAD - Allow file reading
S_IWRITE - Allow file writing

If the values of the above two macros are bitwise OR'ed together, and passed as the **mode** parameter, the file is assumed to allow both reading and writing.

As this function is not supported in the ANSI standard library, its use in new work is discouraged (see the "fopen", "fclose", "fread", and "fwrite" functions for the recommended replacements).

RETURN VALUE

The **creat** function returns a positive value (a "handle"), for the file if the open/create is successful, otherwise (-1) is returned and "errno" is set appropriately.

RELATED FUNCTIONS

close, dup, dup2, open, read, write

ctime

format local time

SYNOPSIS

#include <time.h>

char *ctime(const time_t *timer)

DESCRIPTION

The **ctime** function converts the calendar time specified by **timer** into local time in the form of an ASCII string. A sample of the string representation:

"Sat Aug 15 01:03:51 1972\n\0"

The **ctime** function may be thought of as a functional equivalent for the following expression:

asctime(localtime(timer))

RETURN VALUE

The **ctime** function returns a pointer to the converted string. Note that the string is stored in a static data area which is shared with the "asctime" function. The string will be overwritten upon subsequent calls to either function.

RELATED FUNCTIONS

asctime, gmtime, localtime, time

div

compute integer quotient and remainder

SYNOPSIS

```
#include <stdlib.h>

div_t div(int num, int denom)
```

DESCRIPTION

The **div** function returns the integer quotient and remainder of dividing **num** by **denom**. If the division is inexact, the sign of the quotient will be the same as the algebraic quotient, and the magnitude of the quotient will be the largest integer less than the magnitude of the algebraic quotient. If the result can be represented, then:

num = quotient * **denom** + remainder

RETURN VALUE

The quotient and remainder are returned in a structure of type "div_t", which has two integer members, "quot" and "rem".

RELATED FUNCTIONS

ldiv

dup/dup2

low level file handle duplication

SYNOPSIS

#include <stdio.h>

int dup(int handle)

int dup2(int handle, int handle2)

DESCRIPTION

The **dup** and **dup2** functions assign a second handle to a function which already has one.

The **dup** function returns the next available handle for the file specified by **handle**.

The dup2 function forces **handle2** to refer to the same file as **handle**. This will require closing any existing file which is opened with **handle2**.

These functions are not supported in the ANSI standard library, thus use in new work is discouraged (see the "fopen", "fclose", "fread", and "fwrite" functions for the recommended replacements).

RETURN VALUE

Both **dup** and **dup2** return (-1) on failure and set the value of "errno" as appropriate. On success, **dup** returns the second handle, **dup2** returns zero.

RELATED FUNCTIONS

close, creat, open, read, write

exit

terminate program execution

SYNOPSIS

```
#include <stdlib.h>

void exit(int status)
```

DESCRIPTION

The **exit** function terminates program execution and returns control to the host environment. The **status** parameter is used to return a exit code for the program.

Other than the **status** code, calling **exit** is equivalent to returning from the "main" function:

1. All open output streams are flushed.

2. All open streams and files are closed.

RETURN VALUE

There is (and can be), no return value, since control is passed to the host environment after **exit** is called.

RELATED FUNCTIONS

None

exp/expf

compute exponential

SYNOPSIS

```
#include <math.h>

double exp(double x)

float expf(float x)
```

DESCRIPTION

The **exp** function returns the floating point exponential function of **x**. The **expf** function does the same thing for a single precision argument.

RETURN VALUE

A range error ("errno" is set to ERANGE), occurs if **x** is too large.

RELATED FUNCTIONS

log, log10, pow