Copyright: SiliconGraphics

Porting Applications to IRIX

Ruud van der Pas

SiliconGraphics

European High Performance Computing Team

Copyright: SiliconGraphics

Porting Issues

Porting applications is usually very easy, but in some cases changes need to be made to the code.

- **IRIX 6.x is a 64–bit operating system**
- **Virtual *user* address space is now 40 bits (= 1 TByte)**
- **This means that addresses are no longer 32 bit!**
- **Most data types remain the same (i.e. REAL is 32 bit)**
- **There is still the possibility to stay with the old 32 bit compilers:**

  % setenv SGI_ABI [–32|–64]

- **Compilers support a 32 bit addressing environment as well through the –n32 option or using:**

  % setenv SGI_ABI –n32

Copyright: SiliconGraphics

Porting Issues

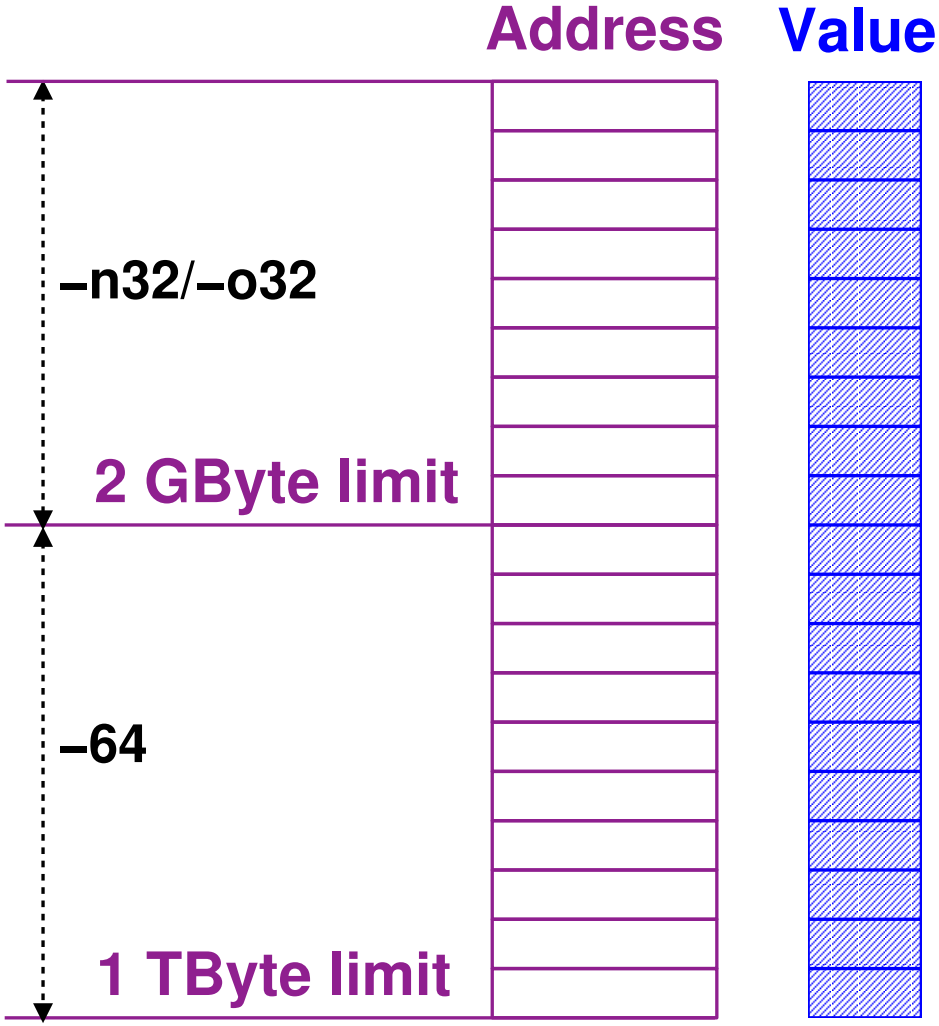
C type	*32–bit*	*64–bit*	*F77 type*
char	8	8	character
short int	16	16	integer*2
int	32	32	integer
long int	32	64	---
long long int	64	64	integer*8
pointer	32	64	pointer
float	32	32	real
double	64	64	real*8
long double	64	128	real*16

red = Potential cause of porting problems

Copyright: SiliconGraphics

Memory – Byte addressable

Do not confuse address and value



Increment in addresses is one byte or more:

char	1 byte
short int	2 bytes
int	4 bytes
long int	4 bytes (–n32)
	8 bytes (–64)
float	4 bytes
double	8 bytes

= 32 bit or 64 bit

= byte (8 bits)

Copyright: SiliconGraphics

Porting Issues

Datatypes start on their "natural" boundary in memory:

C	Fortran	Boundary
char	character	8 bit
int	integer	32 bit
float	real	32 bit
double	double precision	64 bit

Things to watch out for:

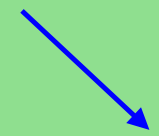
- **Single datatype big common block in Fortran**
- **Structures and unions in C**

In general (!) a C program has more potential porting pitfals than a Fortran program

Copyright: SiliconGraphics

Porting Issues – Fortran Example/1

```
COMMON /WORK/ IARRAY(8)
CALL SUBA(IARRAY(2))
```



```
SUBROUTINE SUBA(X)

DOUBLE PRECISION X(2)
```

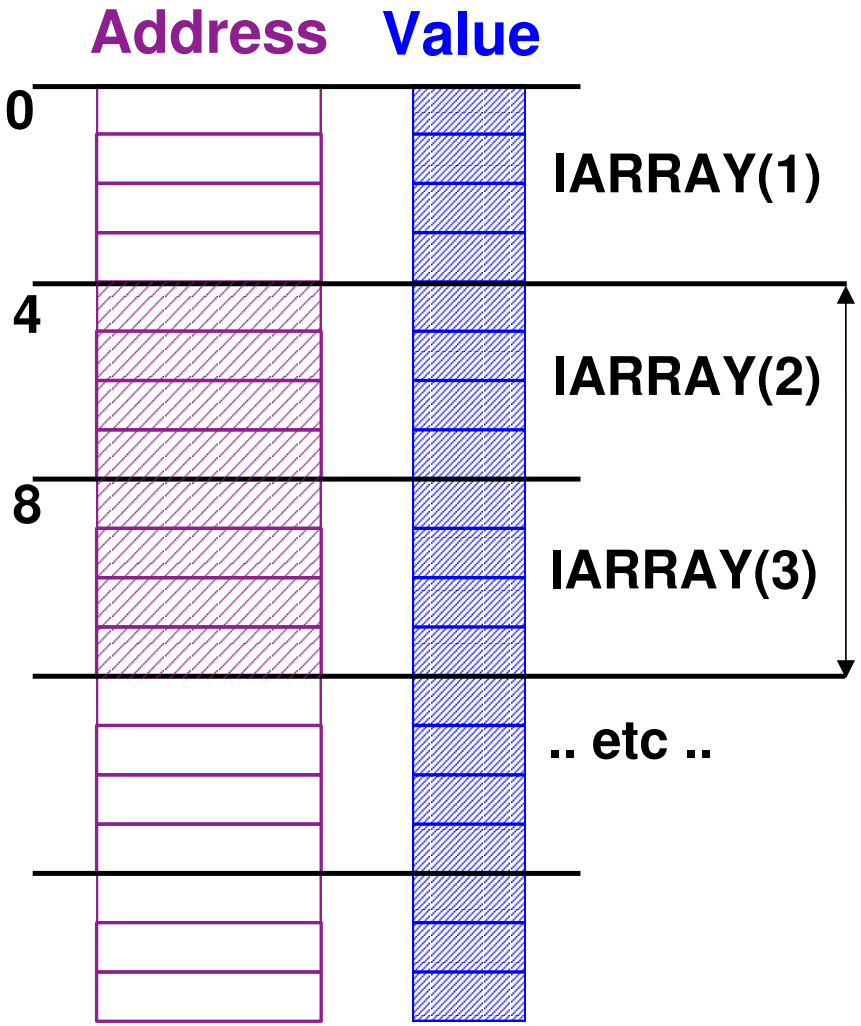
Array X is expected to start on a *64 bit* boundary

However, it is on a 32 bit boundary => bus error

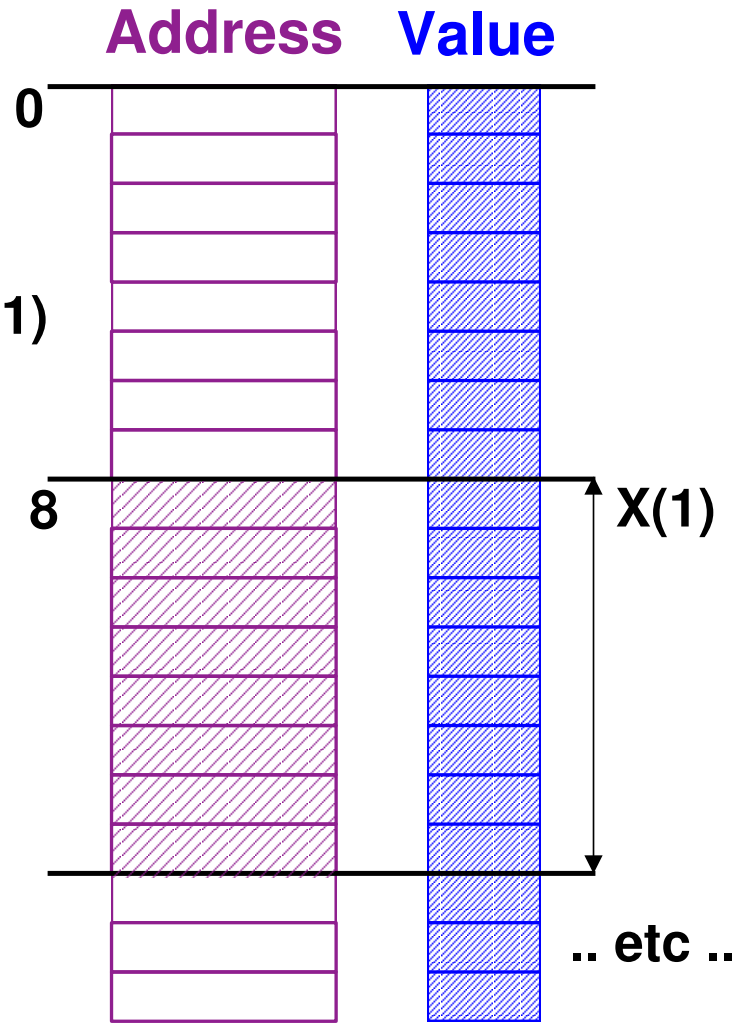
Copyright: SiliconGraphics

Porting Issues – Fortran Example/2

Intended in calling program



Expected within the subroutine



Copyright: SiliconGraphics

The LP64 model

Things to watch out for:

padding: **struct t { char c1,c2; short s; long l;}**

%loc – %val: **these return 64 bit values now (parm. passing)**

sizeof:
```
sizeof(long)   != 4
sizeof(void *) != 4
sizeof(int)    != sizeof(long)
sizeof(long)   != sizeof(float)
```

pointer:
```
int *p,i;
i = (int) p;   /* truncation */
p = (int *) i; /* extension  */
```

constant: **0xffffffff = 0x00000000ffffffff = 4294967295 != –1**

Example Program/1

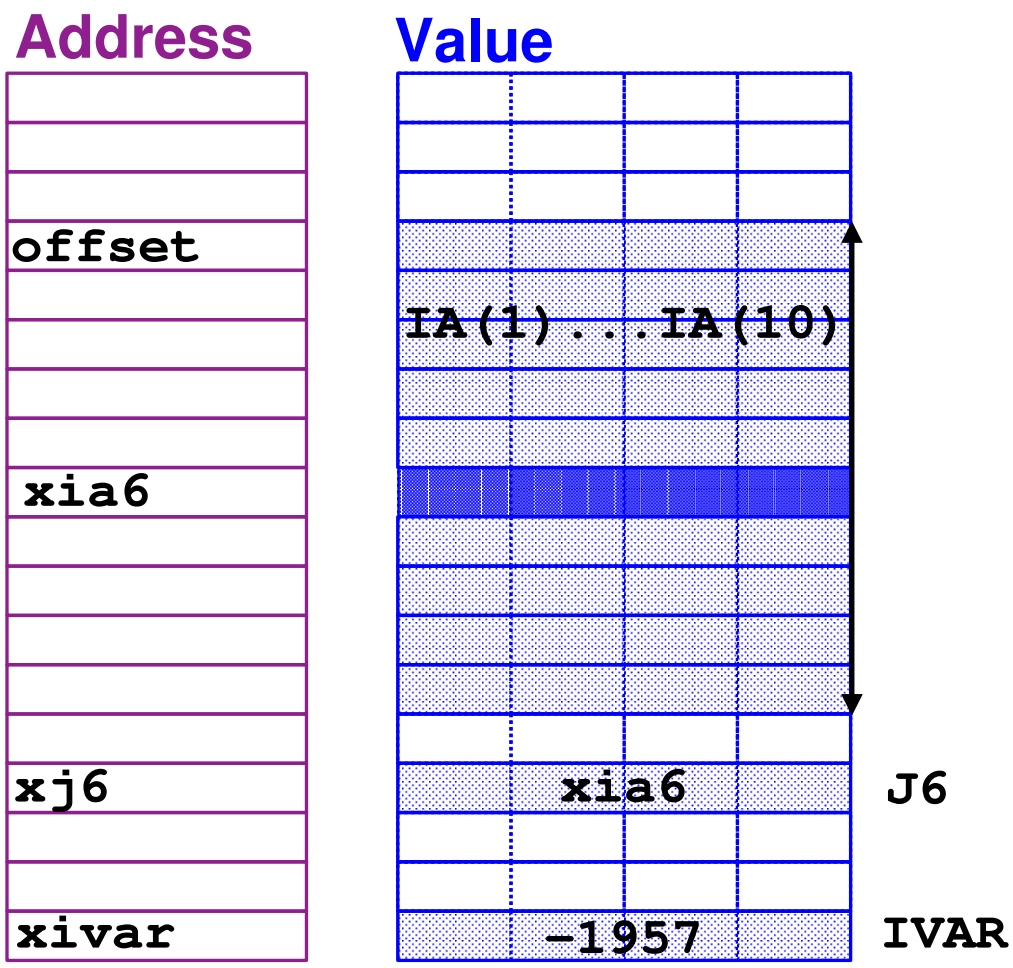
```
program lp64
implicit none

integer offset, j6
integer ivar
integer ia(10)

ia(5) = -1
ia(6) = 12345
ia(7) = 2

print *,'%loc(ia(1))=',%loc(ia(1))

offset = %loc(ia(1))
j6 = offset + 4*(6-1)  ! = %loc(ia(6))

ivar = -1957
call getval (ivar, j6)! j6 = address

print *,'After getval: ivar=',ivar

stop
end
```

```
void getval_ (int *item,
              int **addr)/* address of address */
{
     int *index;

     index = *addr;

     *item = *index;
}
```

The code will run compiled with –n32, but crash when compiled with –64

Copyright: SiliconGraphics

Example Program/2




```
call getval(ivar, j6) :

void getval_ (int *item, int **addr)
{
   int *index;
   index   =  *address;
   *ivar   =  *index;
}
```

Note that:

- Arguments are pointers to ints
- index is a *pointer* to an int variable
- *index = the contents of this pointer

=> Effect of this call:

index = pointer to *contents* of j6 = xia6
*ivar = *index = ia(6) = 12345

Copyright: SiliconGraphics

Example Program/3 – 32 bit: okay

```
        cc -c -n32 funcs.c
        f77 -c -n32 lp64.f
        f77 -o lp64 -n32 funcs.o lp64.o
        ./lp64
Addresses of IA array:
address IA( 1)=  2147429976 value is              0
address IA( 2)=  2147429980 value is      263582800
address IA( 3)=  2147429984 value is              0
address IA( 4)=  2147429988 value is              1
address IA( 5)=  2147429992 value is             -1
address IA( 6)=  2147429996 value is          12345
address IA( 7)=  2147430000 value is              2
address IA( 8)=  2147430004 value is              0
address IA( 9)=  2147430008 value is              0
address IA( 10)=  2147430012 value is             0
Address pointer j6 =   2147429996
Before getval: ivar=       -1957
After  getval: ivar=       12345
```

Copyright: SiliconGraphics

Example Program/4 – 64 bit: problems

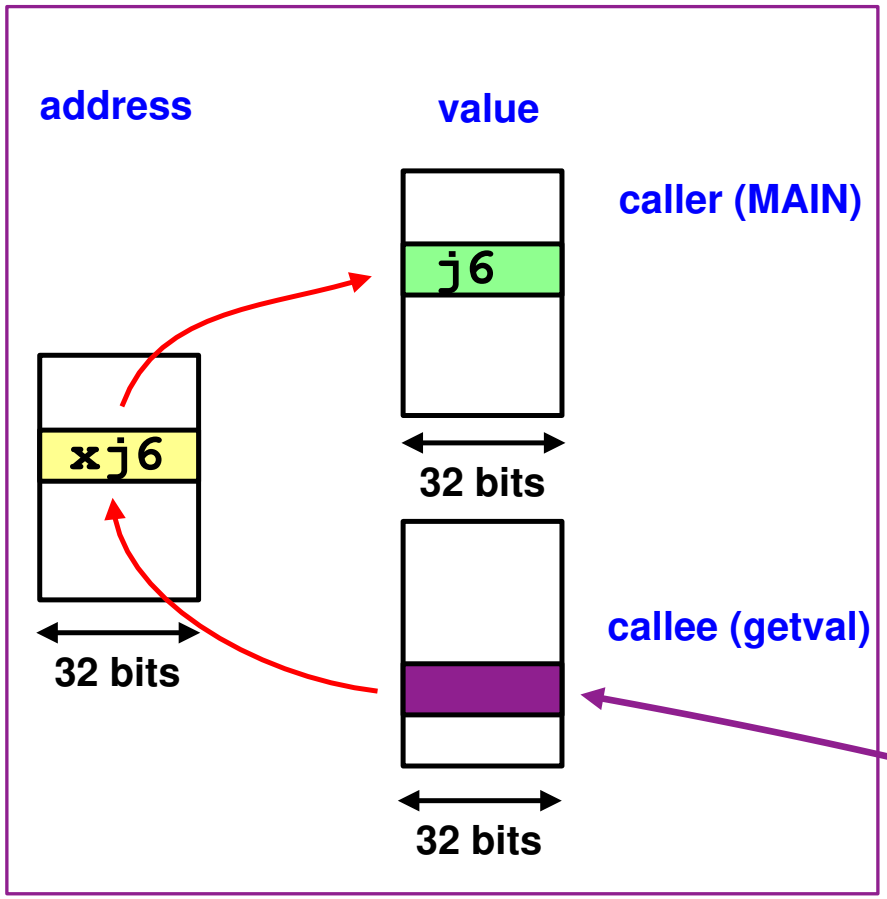
```
        cc -c -64 funcs.c
        f77 -c -64 lp64.f
"lp64.f", line 9: warning(1412): destination type of cast is too small to
hold all pointers: truncation possible
        offset = %loc(ia(1))
                 ^
        f77 -o lp64 -64 funcs.o lp64.o
        ./lp64
 Addresses of IA array:
 address IA( 1)=   1099511606648 value is          255
 address IA( 2)=   1099511606652 value is       -21088
 address IA( 3)=   1099511606656 value is            0
 address IA( 4)=   1099511606660 value is           15
 address IA( 5)=   1099511606664 value is           -1
 address IA( 6)=   1099511606668 value is        12345
 address IA( 7)=   1099511606672 value is            2
 address IA( 8)=   1099511606676 value is    263606220
 address IA( 9)=   1099511606680 value is          255
 address IA(10)=   1099511606684 value is       -20840
 Address pointer j6 =      -21108
 Before getval: ivar=      -1957
Bus error (core dumped)
```

uninitialized

uninitialized

Copyright: SiliconGraphics

Example Program/5 – Code #1

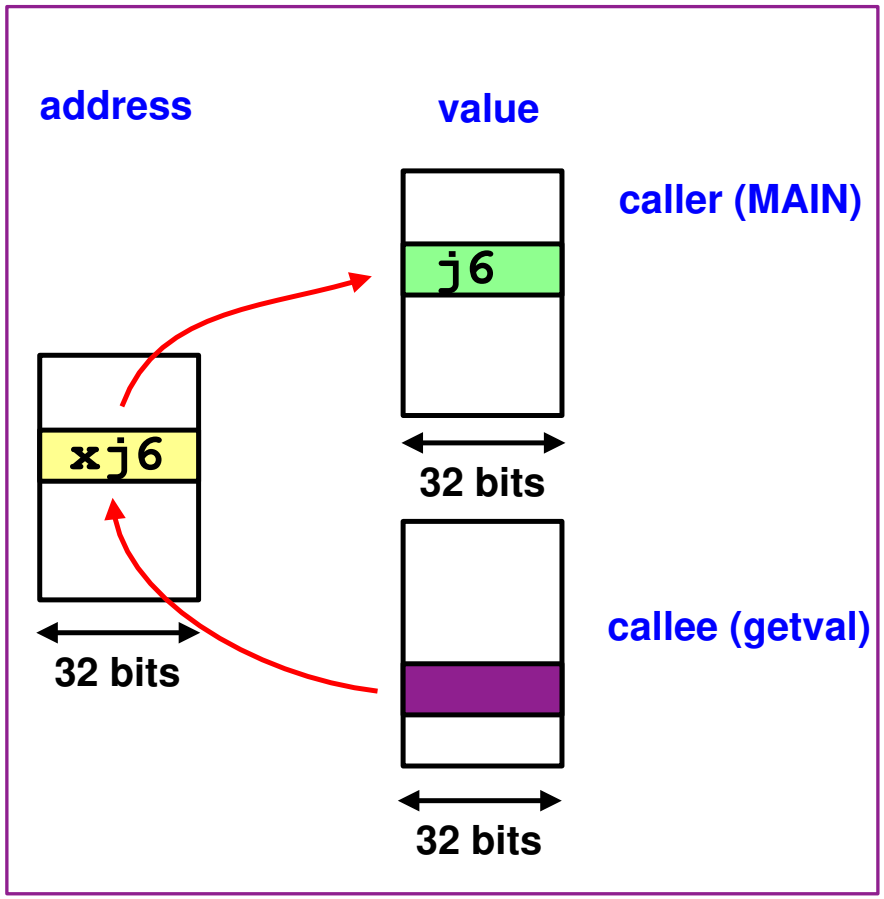


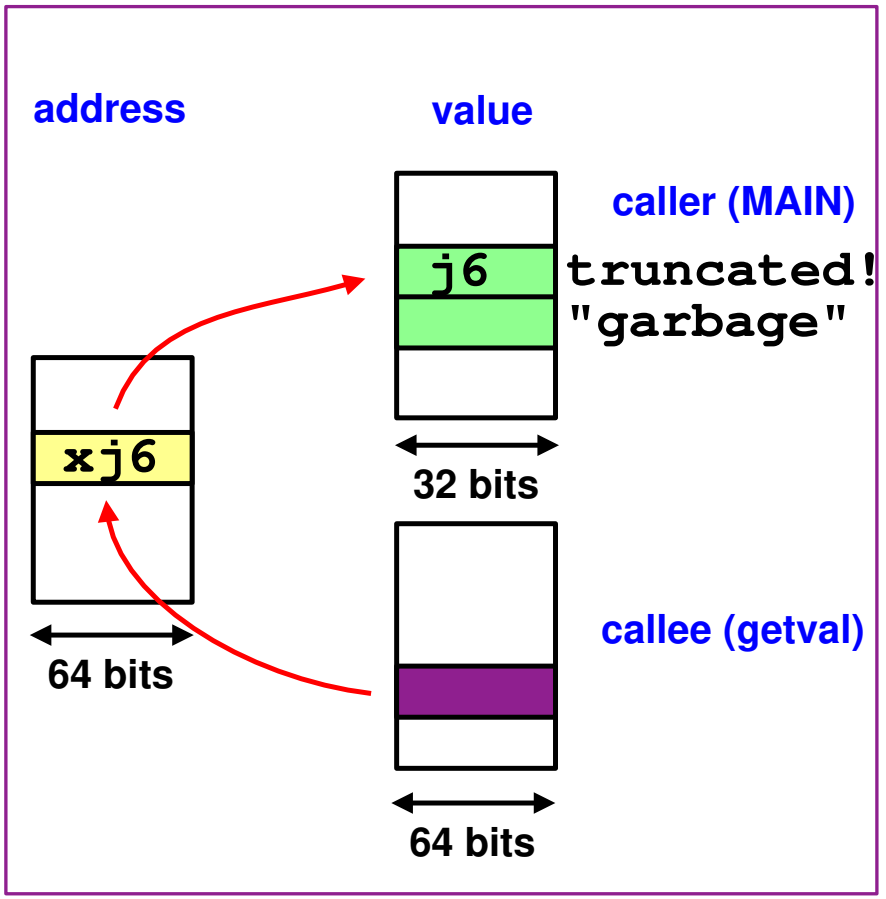
**addr is the address of an address

-n32

Copyright: SiliconGraphics

Example Program/6 – Code #1





–n32

–64

2 errors: truncated+garbage

Copyright: SiliconGraphics

Example Program/7

```
program lp64
implicit none

integer*8 offset, j6
integer ivar
integer ia(10)

ia(5) = -1
ia(6) = 12345
ia(7) = 2

print *,'%loc(ia(1))=',%loc(ia(1))

offset = %loc(ia(1))
j6 = offset + 4*(6-1)  ! = %loc(ia(6))

ivar = -1957
call getval (ivar, j6)! j6 = address

print *,'After getval: ivar=',ivar

stop
end
```

```
void getval_ (int *item,
              int **addr)/* address of address */
{
     int *index;

     index = *addr;

     *item = *index;
}
```

The code will now run compiled with –64, but crash when compiled with –n32 .....

Copyright: SiliconGraphics

Example Program/8 – 64 bit: okay

```
cc -c -64 funcs.c
f77 -c -64 lp64.f
f77 -o lp64 -64 funcs.o lp64.o
./lp64
Addresses of IA array:
address IA( 1)=  1099511606632 value is                0
address IA( 2)=  1099511606636 value is                0
address IA( 3)=  1099511606640 value is                0
address IA( 4)=  1099511606644 value is        229914144
address IA( 5)=  1099511606648 value is               -1
address IA( 6)=  1099511606652 value is            12345
address IA( 7)=  1099511606656 value is                2
address IA( 8)=  1099511606660 value is               15
address IA( 9)=  1099511606664 value is                0
address IA(10)=  1099511606668 value is        229914144
Address pointer j6 =    1099511606652
Before getval: ivar=     -1957
After  getval: ivar=     12345
```

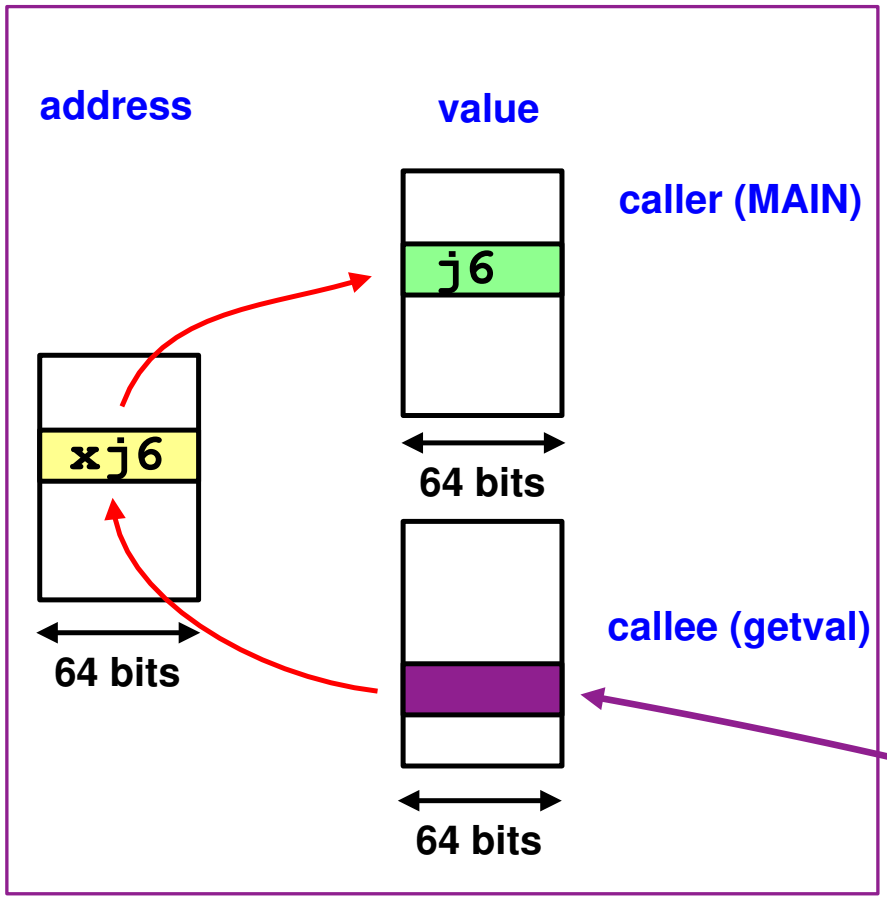
Example Program/9 – 32 bit: problems

```
cc -c -n32 funcs.c
f77 -c -n32 lp64.f
f77 -o lp64 -n32 funcs.o lp64.o
./lp64
 Addresses of IA array:
 address IA( 1)=  2147429976 value is              0
 address IA( 2)=  2147429980 value is      263582800
 address IA( 3)=  2147429984 value is              0
 address IA( 4)=  2147429988 value is              1
 address IA( 5)=  2147429992 value is             -1
 address IA( 6)=  2147429996 value is          12345
 address IA( 7)=  2147430000 value is              2
 address IA( 8)=  2147430004 value is              0
 address IA( 9)=  2147430008 value is              0
 address IA(10)=  2147430012 value is              0
 Address pointer j6 =              2147429996
 Before getval: ivar=         -1957
*item = -1957
*addr = 0
index  = 0
Segmentation fault (core dumped)
```

Argument j6 is integer*8
With –n32, a long int is 32 bit ...

Copyright: SiliconGraphics

Example Program/10 – Code #2

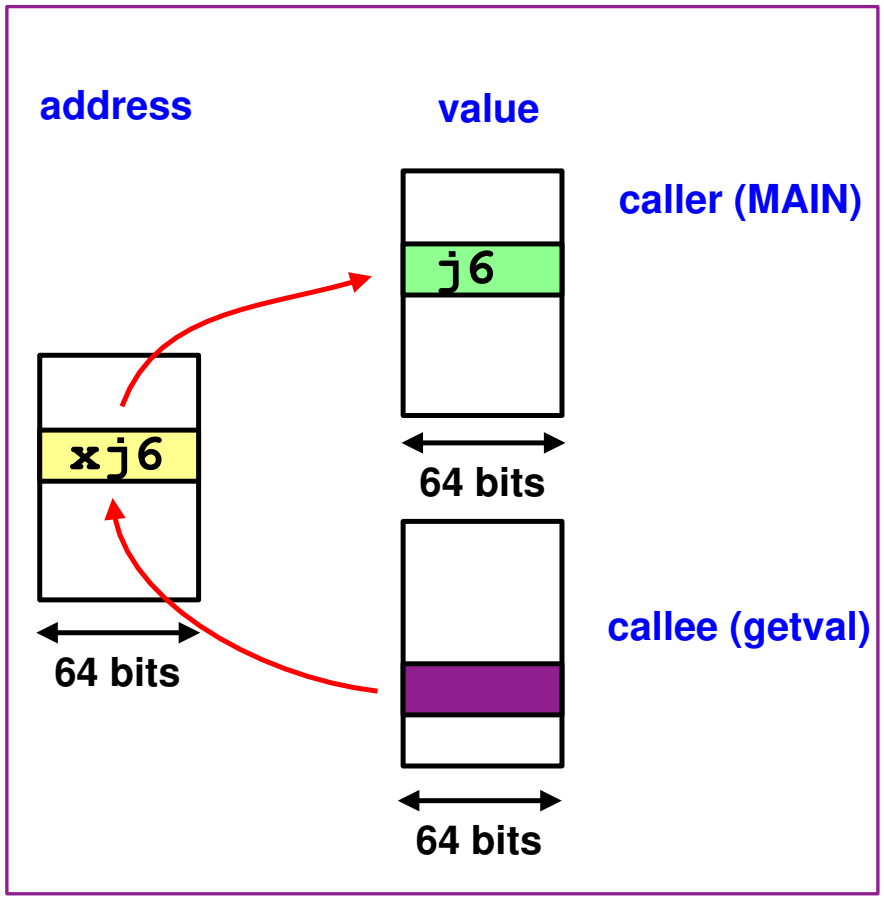


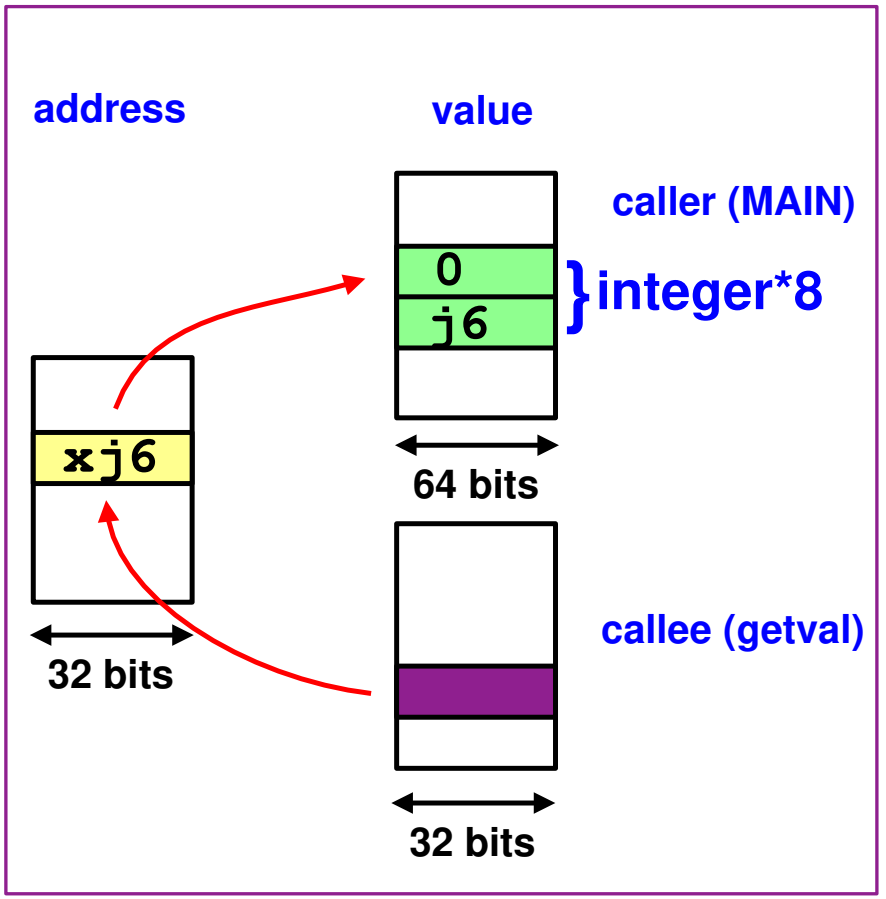
**addr is the address of an address

–64

Copyright: SiliconGraphics

Example Program/11 – Code #2





–64

–n32

1 error: picks up 0 as address

Copyright: SiliconGraphics

Macros

Note that certain macros are set. For example:

Macro	32–bit	64–bit
–MIPS_SZINT	32	32
–MIPS_SZLONG	32	64
–MIPS_SZPTR	32	64

For much more information: `% man abi`

Copyright: SiliconGraphics

Summary 32–64 bit

- *64–bit addressing and full performance*

  use –64 –mips4

- *32–bit addressing and full performance*

  use –n32 –mips4

- *32–bit addressing and less performance*

  use –o32

Static versus automatic storage

What should be the printed value of abc in the second call?

Static storage: *1*

Local variables maintain values between the calls

Automatic storage: *undetermined*

Local variables do not maintain values between the calls

```
program test

call suba(1)
call suba(2)

stop
end

subroutine suba(i)

integer i, abc

print *,'abc = ',abc
abc = i
return
end
```

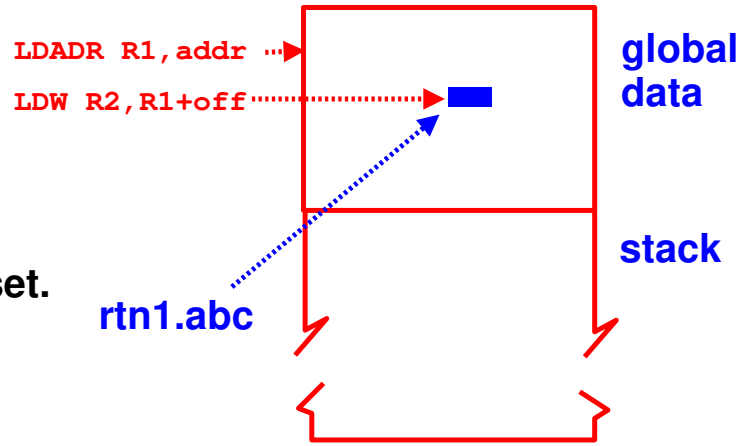
Copyright: SiliconGraphics

Static versus automatic storage

Static

* As offsets to pointers are limited to 64KB, first load a register with a base address then load the data with an offset.

* Permanent storage of local (temporary) variables



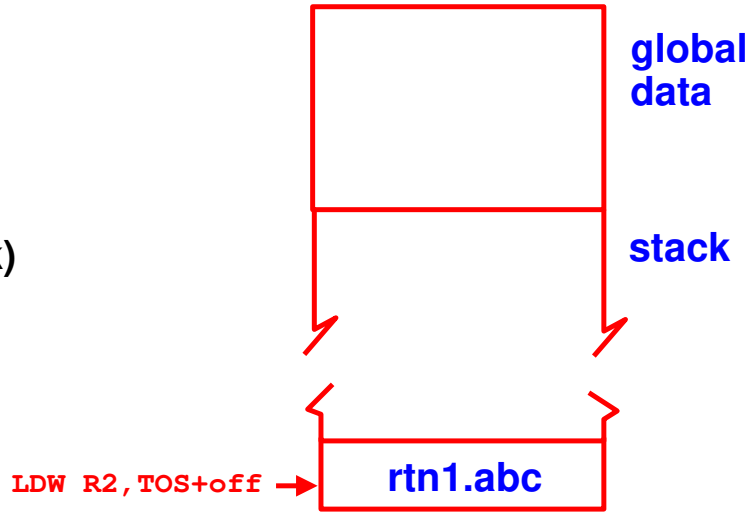
Copyright: SiliconGraphics

Static versus automatic storage

Automatic

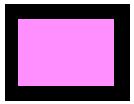
* One operation to fetch a word of memory using the TOS (Top Of Stack) pointer

* Temporary storage of local (temporary) variables



Copyright: SiliconGraphics

Floating Point Exceptions

Interested?

Click here .....