我是軟體
那些處理器教我的事

Jim Huang(黃敬群) “jserv”
website: http://jserv.sayya.org/
blog: http://blog.linux.org.tw/jserv/

COSCUP – Aug 24, 2008

猜猜看

軟體若是女人，胸圍該落在哪個範圍？

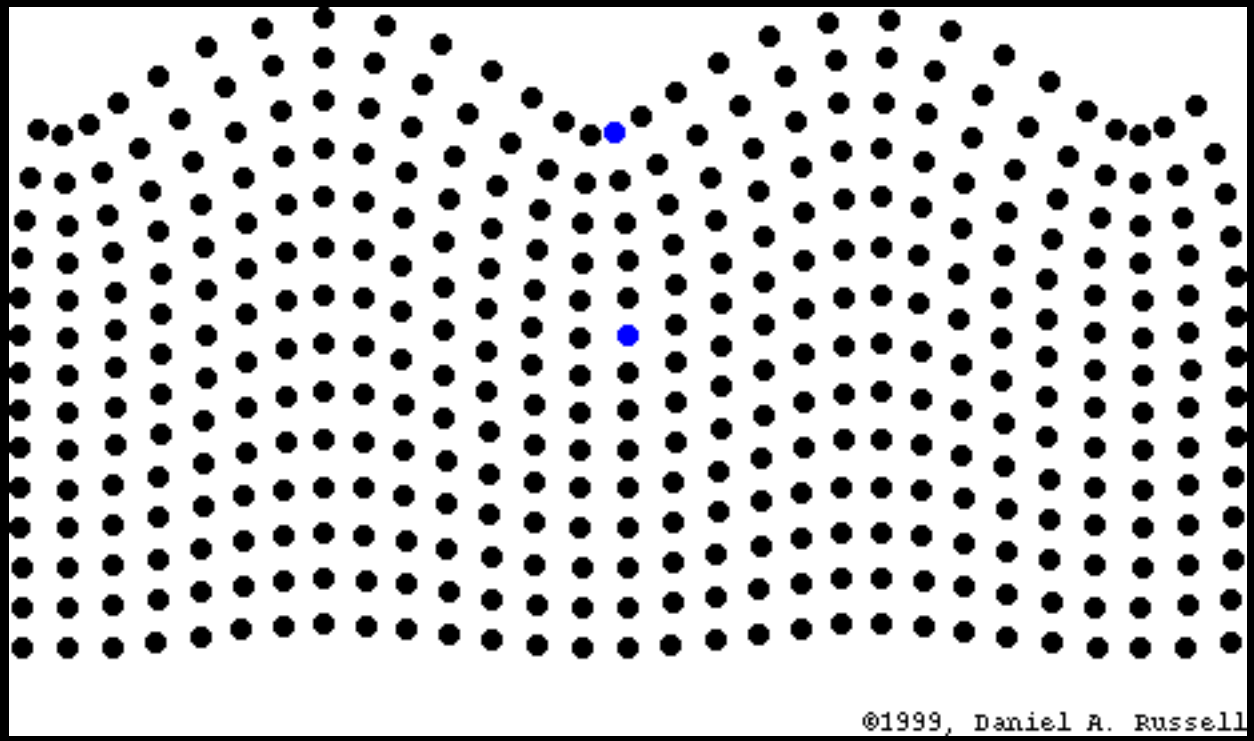
令 x, y 表示運動軌跡

x= x0 + r * **cos** (phi)

y= y0 + r * sin (phi)

phi= 2π/λ+ 2π/ T

λ爲波長 T 爲週期 r 爲振幅



海波浪

女人是水做的

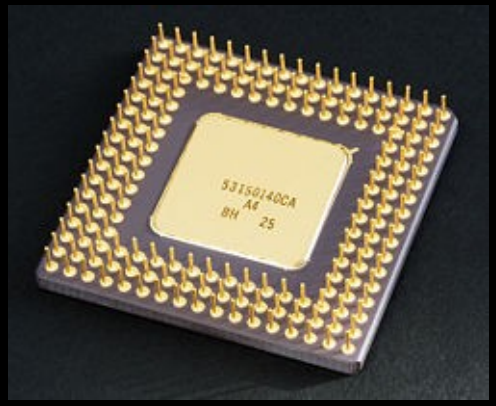

是的，就是 cos 罩杯
COSCUP!

從 CUP 到 CPU

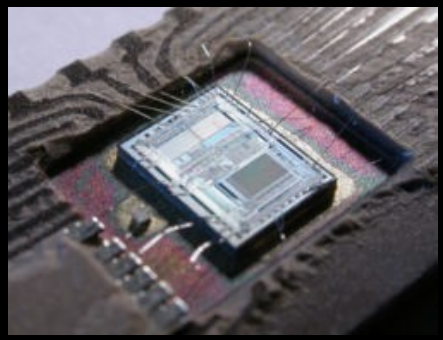
CPU/ 中央處理器

MOS 6502

Intel 80486DX2



intel 8742



AMD, ARM, Freescale, IBM, PowerPC, x86, MIPS, Sun/SPARC, VIA, ..., 龍芯, 台灣心

三折肱成良醫

軟體被處理器玩弄／調教的經驗

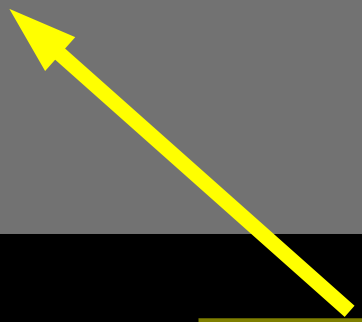
迷思

C 語言的程式碼可輕易移植到各種平台

處理器想的跟你不一樣 (1)

- alignment.c

```
#include <stdio.h>
int main () {
  char *str = "\x01\x23\x45\x67\x89\xab\xcd\xef";
  unsigned *u = (unsigned *)(str + 1);
  printf ( "%08x\n", *u );
  return 0;
}
```



印列前 8 個 16 進位字元並換行

預期執行輸出

- x86

```
jserv@venux:~$ 486-linux-gnu-gcc -o
  alignment.x86 alignment.c
jserv@venux:~$ ./alignment.x86
89674523
```

- arm

```
jserv@venux:~$ arm-angstrom-linux-gnueabi-gcc
  -o alignment.arm alignment.c
root@om-gta02:~# ./alignment.arm
89674523
```

實際執行輸出

- arm

```
root@om-gta02:~# echo 0 > /proc/cpu/alignment
root@om-gta02:~# ./alignment.arm
01674523
root@om-gta02:~# echo 1 > /proc/cpu/alignment
root@om-gta02:~# ./alignment.arm
01674523
root@om-gta02:~# echo 2 > /proc/cpu/alignment
root@om-gta02:~# ./alignment.arm
89674523
root@om-gta02:~#
```

「慣 C」玩弄軟體無數，也有出包時

「我好天真，現在才看清，其實這一切並非只是 cross compile 這麼單純」

```
root@om-gta02:~# echo 0 > /proc/cpu/alignment
root@om-gta02:~# cat /proc/cpu/alignment
User:       9
System:     0
Skipped:    0
Half:       0
Word:       5
Multi:      0
User faults:    0 (ignored)
```

```
root@om-gta02:~# echo 3 > /proc/cpu/alignment
root@om-gta02:~# cat /proc/cpu/alignment
User:       9
System:     0
Skipped:    0
Half:       0
Word:       5
Multi:      0
User faults:    3 (fixup+warn)
```

```
root@om-gta02:~# echo 1 > /proc/cpu/alignment
root@om-gta02:~# cat /proc/cpu/alignment
User:       9
System:     0
Skipped:    0
Half:       0
Word:       5
Multi:      0
User faults:    1 (warn)
```

```
root@om-gta02:~# echo 2 > /proc/cpu/alignment
root@om-gta02:~# cat /proc/cpu/alignment
User:       9
System:     0
Skipped:    0
Half:       0
Word:       5
Multi:      0
User faults:    2 (fixup)
```

在非 x86 的平台上，
2 bytes 或 4 bytes 長度
的整數變數表示方式
大相逕庭
（是否爲偶數整除）

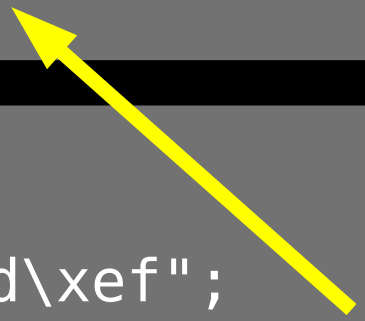
```
char *str = "\x01\x23\x45\x67\x89\xab\xcd\xef";
```

aligned address 與否

```
    +3 +2 +1 +0
    67 45 23 01  字串表示法 (Litten Endian)
    ef cd ab 89

    +3 +2 +1 +0
    02 01 00     x86 access to
             03 (unsigned value is access to positions of bytes)

    +3 +2 +1 +0
    02 01 00 03 ARM, which is accessed (as above)
```



```
#include <stdio.h>
int main () {
  char *str = "\x01\x23\x45\x67\x89\xab\xcd\xef";
  unsigned *u = (unsigned *)(str + 1);
  printf ( "%08x\n", *u );
  return 0;
}
```

在 ARM/Linux 上，/proc/cpu/alignment 可更改此類 exception 的行爲（可透過 dmesg 觀察）

```
root@om-gta02:~# echo 0 > /proc/cpu/alignment
root@om-gta02:~# cat /proc/cpu/alignment
User:       9
System:     0
Skipped:    0
Half:       0
Word:       5
Multi:      0
User faults:    0 (ignored)
```

```
root@om-gta02:~# echo 2 > /proc/cpu/alignment
root@om-gta02:~# cat /proc/cpu/alignment
User:       9
System:     0
Skipped:    0
Half:       0
Word:       5
Multi:      0
User faults:    2 (fixup)
```

aligned
read →…
x86 unaligned
read → read →…
ARM unaligned (fixup)
read → (trap) → emulate → (return) →…

```
#include <stdio.h>
int main () {
  char *str = "\x01\x23\x45\x67\x89\xab\xcd\xef";
  unsigned *u = (unsigned *)(str + 1);
  printf ( "%08x\n", *u );
  return 0;
}
```

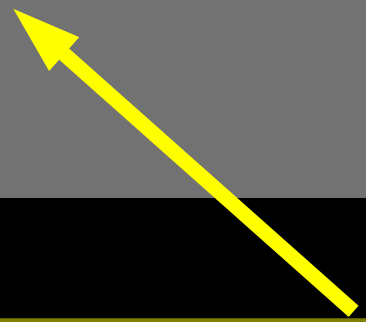
C 語言的程式碼可輕易移植到各種平台？

注意 Alignment

處理器想的跟你不一樣 (2)

- signchar.c

```
#include <stdio.h>
int main()
{
  char c1 = 0xff, c2 = 0x01;
  printf((c1 > c2) ? "c1 > c2\n" : "c1 <= c2\n");
  return 0;
}
```



就只是比大小，補數問題
書本：「char 的範圍是從 CHAR_MIN 到 CHAR_MAX，所以是 -128 到 +127」

實際執行輸出

- x86

```
jserv@venux:~$ i486-linux-gnu-gcc -o
  signchar.x86 signchar.c
jserv@venux:~$ ./signchar.x86
c1 <= c2
```

- arm

```
jserv@venux:~$ arm-angstrom-linux-gnueabi-gcc
  -o signchar.arm signchar.c
root@om-gta02:~# ./signchar.arm
c1 > c2
```

char是乾的不能洗；
char是乾淨的，沒有髒也不需要洗

「現在重修C語言，來得及嗎？」

signchar.c

```
#include <stdio.h>
int main()
{
  char c1 = 0xff, c2 = 0x01;
  printf((c1 > c2) ? "c1 > c2\n" : "c1 <= c2\n");
  return 0;
}
```

這個 char 跟阿扁的錢一樣神奇，嘖嘖
「阿扁錯了嗎？」

(ARM Toolchain 特有)
以 char 宣告的 unsigned char 數值
→「被視作 unsigigned char 變數」
實際上，此行爲在 C 語言的規範，乃是「與實做相關」

-funsigned-char

Let the type "char" be unsigned, like "unsigned char".

Each kind of machine has a default for what "char" should be. It is either like "unsigned char" by default or like "signed char" by default.

Ideally, a portable program should always use "signed char" or "unsigned char" when it depends on the signedness of an object. But many programs have been written to use plain "char" and expect it to be signed, or expect it to be unsigned, depending on the machines they were written for. This option, and its inverse, let you make such a program work with the opposite default.

The type "char" is always a distinct type from each of "signed char" or "unsigned char", even though its behavior is always just like one of those two.

char 似你的温柔

-fsigned-char

Let the type "char" be signed, like "signed char".

Note that this is equivalent to -fno-unsigned-char, which is the negative form of -funsigned-char. Likewise, the option -fno-signed-char is equivalent to -funsigned-char.

```
jserv@venux:~$ arm-angstrom-linux-gnueabi-gcc -o
  signchar-2.arm -fsigned-char signchar.c
root@om-gta02:~# ./signchar-2.arm
c1 <= c2
```

C 語言的程式碼可輕易移植到各種平台？

注意平台實做相依議題

處理器想的跟你不一樣 (3)

- abi.c

```
#include <stdio.h>
struct foo {
  char a;
};
int main()
{
  printf("sizeof(struct foo) = %d\n",
  sizeof(struct foo));
  return 0;
}
```

實際執行輸出

- x86

```
jserv@venux:~$ i486-linux-gnu-gcc -o abi.x86
  abi.c
jserv@venux:~$ ./abi.x86
sizeof(struct foo) = 1
```

- arm(EABI)

```
jserv@venux:~$ arm-angstrom-linux-gnueabi-gcc -o
  abi.arm abi.c
root@om-gta02:~# ./abi.arm
sizeof(struct foo) = 1
```

- arm(OABI)

```
jserv@venux:~$ arm-linux-gcc -o abi-oabi.arm
  -static abi.c
root@om-gta02:~# ./abi-oabi.arm
sizeof(struct foo) = 4
```

平平都在 ARM 上面跑，怎麼會有差？

abi.c

```
#include <stdio.h>
struct foo {
  char a;
};
int main()
{
  printf("sizeof(struct foo) = %d\n", sizeof(struct foo));
  return 0;
}
```

針對 C 語言未能涵蓋的部份，**ABI** (Application Binary Interface) 規範以下實做表現：

- 宣告元素組成的 structure
- 函式呼叫
  - 函式參數傳遞、暫存器使用，以及如何進入與離開函式
  - 對 stack 的操作
  - stack pointer 的執行時期行爲

- legacy ABI / OABI (Old ABI)
  -mabi=apcs-gnu
- ARM EABI
  -mabi=aapcs-linux

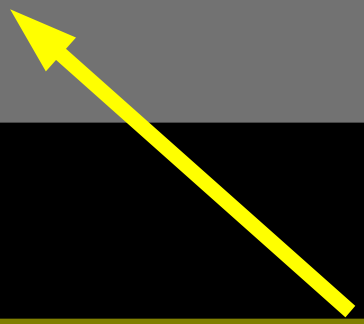
C 語言的程式碼可輕易移植到各種平台？

注意 ABI 表現

處理器想的跟你不一樣 (4)

- call.c

```
#include <string.h>
int foo(char *to, char *from)
{
        char buf[1024];
        memcpy(buf, "12345", 6);
}
```



透過 binutils 裡面的 nm 工具程式，可觀察到編譯後輸出的符號

實際分析輸出

- x86

```
jserv@venux:~$ i486-linux-gnu-gcc -O0 -c -o
  call-x86.o call.c
jserv@venux:~$ nm call-x86.o
         U __stack_chk_fail
00000000 T foo
         U memcpy
```

- arm

```
jserv@venux:~$ arm-angstrom-linux-gnueabi-gcc
  -O0 -c -o call-arm.o call.c
jserv@venux:~$ arm-angstrom-linux-gnueabi-nm
  call-arm.o
00000000 T foo
```

符號走丟了嗎？

- call.c

```
#include <string.h>
int foo(char *to, char *from)
{
        char buf[1024];
        memcpy(buf, "12345", 6);
}
```

透過 binutils 裡面的 nm 工具程式，可觀察到編譯後輸出的符號

memcpy, memmove, memset, strcpy, stpcpy, strncpy, strcat, strncat 等函式會直接展開爲 ARM 平台優化的機械碼

[注意] 內部記憶體位址

- arm

```
jserv@venux:~$ arm-angstrom-linux-gnueabi-gcc -O0 -c -o
  call-arm.o call.c
jserv@venux:~$ arm-angstrom-linux-gnueabi-nm call-arm.o
00000000 T foo
```

C 語言的程式碼可輕易移植到各種平台？

符號處理可能會變動

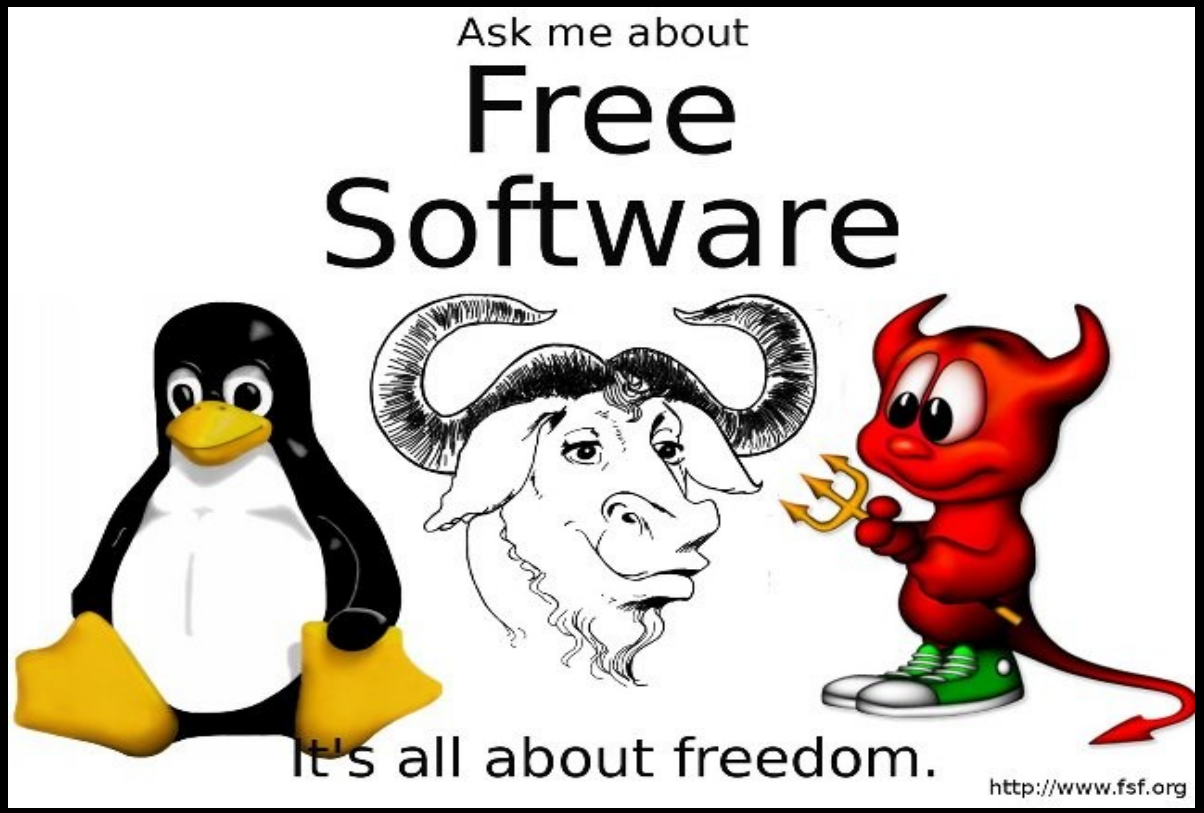
女人要男人回答的不是
「眞實」的答案，而是
「正確」的答案

《我是女王2-- 那些壞男人教我的事》

「正確」的答案

考慮的議題：

alignment, ABI, packed data, char signed, soft/hard floating-point, data abort exception, implementor, cache/TLB, ...



打破非技術的藩籬，允許在不同的軟硬體環境中，自由的執行與改良軟體，就是自由軟體最大的價值

→理解不同處理器的特性
對自由軟體的廣泛應用
非常重要

結語

- 處處留心皆學問！
- 諸多自由軟體雖如 jjhou 所言：「源碼之前了無秘密」，但往往僅在主流平台如 x86 與 PowerPC 上開發測試
- 要運用於 ARM 或 MIPS 一類的嵌入式系統大宗的微處理器時，需要考量硬體的差異、軟體規劃的歧異性
- 自由軟體基於開放發展，已慢慢由多元的處理器所「調校」與「調教」，質與量的精進可期

參考資訊

- ARM-EABI 官方資訊
  http://www.arm.com/products/DevTools/ABI.html
- ARM Linux 開發總部

  http://www.arm.linux.org.uk/
- 用 Open Source 工具開發軟體

  http://www.study-area.org/cyril/opentools/opentools/book1.html

廣告！

- 休假了近八個月的長假，因爲太無聊，又撰寫新的作業系統
- 定位於 RTOS 開發的實例 (COSCUP 2007 發表的精簡版本)
- 僅 20kb ，針對 ARM/Xscale 平台

CuRT - Compat Unicellular Real-Time operating system

- 特性：
  - Preemptive Multi-threading
  - Priority-base Round-Robin Scheduling
  - Thread Management
  - Semaphore Management Support
  - IPC: mailbox, message queue
- 支援硬體： Gumstix connex (based on PXA255, armv5te)
- BSD License