Inline Assembly

Overview

- What is inline assembly
- How it is done
- What can be gained by it
- What can be lost
- Sample

What is Inline Assembly?

- A method of incorporating assembly language directly into a C program
- Bypasses the compiler

How it is Done

- Basic inline assembly
  - *asm (”statements”);* where statements consists of one or more assembly commands

```
/* branch to the function f1() */
asm (“b f1”);
```

How it is Done

- Extended inline assembly
  - *asm ("statements":output registers:input registers:changed registers);*
  - *__asm__* if asm is a keyword
  - *asm volatile ()* if the statements must execute where you put them
    - Optimization may move statements otherwise

What is Gained with Inline Assembly

- Decreased overhead for function calls
- More direct control of the program
- SPEED!

What May be Lost

- Program size may increase
- Ease of debugging
- Time spent learning assembly commands
- If -O3 optimization is already used, speed increase may be negligible

An Example of Inline Assembly

- A useful example: *memcpy()*
- What's wrong with it?
- How to fix it
- Inlining *memcpy()*

A useful example: memcpy()

- A standard C function
- Copies memory byte by byte from one location to another
- Not written with the R3000 in mind

What's wrong with it

- It's very inefficient
  - With the R3000, reading a byte takes as much time as reading a word
    - Number of reads is 4X what it needs to be
  - The write buffer holds four words, with successive writes giving faster access
    - Writing immediately after a read fails to take advantage of this

How to fix it

- Lessen the number of reads
  - Make a version of *memcpy()* that reads words instead of bytes at a time
- It should also utilize the speed bonus gained by with successive writes
  - Change the RWRWRWRW… to RRRRWWWW…

Inlining memcpy()

- Getting started
  - Look at *memcpy()* in the disassembly window of the debugger
  - Delete unneeded commands
  - Add a label or two
  - Copy it into an assembler macro

Inlining memcpy()

- *memcpy()* disassembled

```
memcpy  >1080000A       beq     a0,zero,$8001052c
80010504 00001021       move    v0,zero
80010508 18C00007       blez    a2,$80010528
8001050C 00801821       move    v1,a0
80010510 90A20000       lbu     v0,0(a1)
80010514 24A50001       addiu   a1,a1,$1
80010518 24C6FFFF       addiu   a2,a2,-$1
8001051C A0820000       sb      v0,0(a0)
80010520 1CC0FFFB       bgtz    a2,$80010510
80010524 24840001       addiu   a0,a0,$1
80010528 00601021       move    v0,v1
8001052C 03E00008       jr      ra
```

Inlining memcpy()

- Unnecessary commands

```
memcpy  >1080000A   beq     a0,zero,$8001052c
80010504 00001021   move    v0,zero
80010508 18C00007   blez    a2,$80010528
8001050C 00801821   move    v1,a0
80010510 90A20000   lbu     v0,0(a1)
80010514 24A50001   addiu   a1,a1,$1
80010518 24C6FFFF   addiu   a2,a2,-$1
8001051C A0820000   sb      v0,0(a0)
80010520 1CC0FFFB   bgtz    a2,$80010510
80010524 24840001   addiu   a0,a0,$1
80010528 00601021   move    v0,v1
8001052C 03E00008   jr      ra
```

Inlining memcpy()

- Add a label, remove addresses, change register names and immediates

```
byteloop:
lbu     $2,0($5)
addiu   $5,$5,1
addiu   $6,$6,-1
sb      $2,0($4)
bgtz    $6,byteloop
addiu   $4,$4,1
```

Inlining memcpy()

- Making a macro, loading arguments

```
#define inlinemcpy(p1, p2, nbytes){ \
        asm ("move    $4,%0": :"r"(p1):"$2","$4","$5","$6","memory"); \
        asm ("move    $5,%0": :"r"(p2):"$2","$4","$5","$6","memory"); \
        asm ("move    $6,%0": :"r"(bytes):"$2","$4","$5","$6","memory"); \
        asm ("byteloop:": : :"$2","$4","$5","$6","memory"); \
        asm ("lbu     $2,0($5)": : :"$2","$4","$5","$6","memory"); \
        asm ("addiu   $5,$5,1": : :"$2","$4","$5","$6","memory"); \
        asm ("addiu   $6,$6,-1": : :"$2","$4","$5","$6","memory"); \
        asm ("sb      $2,0($4)": : :"$2","$4","$5","$6","memory"); \
        asm ("bgtz    $6,byteloop": : :"$2","$4","$5","$6","memory"); \
        asm ("addiu   $4,$4,1": : :"$2","$4","$5","$6","memory"); \
}
```

Inlining memcpy()

- Change the byte commands to word commands
- The byte version will still be used for “leftovers”
  - Assuming you don’t want limit yourself to word-length transfers

Inlining memcpy()

- Word version

argument loading

```
:
asm ("wordloop:": : :"$2","$4","$5","$6","memory"); \
asm ("lw      $2,0($5)": : :"$2","$4","$5","$6","memory"); \
asm ("addiu   $5,$5,4": : :"$2","$4","$5","$6","memory"); \
asm ("addiu   $6,$6,-4": : :"$2","$4","$5","$6","memory"); \
asm ("sw      $2,0($4)": : :"$2","$4","$5","$6","memory"); \
asm ("bgtz    $6,wordloop": : :"$2","$4","$5","$6","memory"); \
asm ("addiu   $4,$4,4": : :"$2","$4","$5","$6","memory"); \
:
```

byte version

Inlining memcpy()

- This version still uses the RWRWRW sequence
- Even faster is four reads followed by four writes

Inlining memcpy()

- Add three more reads and writes

```
asm ("wordloop:": : :"$2","$4","$5","$6","memory"); \
asm ("lw      $8,0($5)": : :"$2","$4","$5","$6","memory"); \
asm ("lw      $9,4($5)": : :"$2","$4","$5","$6","memory"); \
asm ("lw      $10,8($5)": : :"$2","$4","$5","$6","memory"); \
asm ("lw      $11,12($5)": : :"$2","$4","$5","$6","memory"); \
asm ("addiu   $5,$5,16": : :"$2","$4","$5","$6","memory"); \
asm ("addiu   $6,$6,-16": : :"$2","$4","$5","$6","memory"); \
asm ("sw      $8,0($4)": : :"$2","$4","$5","$6","memory"); \
asm ("sw      $9,4($4)": : :"$2","$4","$5","$6","memory"); \
asm ("sw      $10,8($4)": : :"$2","$4","$5","$6","memory"); \
asm ("sw      $11,12($4)": : :"$2","$4","$5","$6","memory"); \
asm ("bgtz    $6,wordloop": : :"$2","$4","$5","$6","memory"); \
asm ("addiu   $4,$4,16": : :"$2","$4","$5","$6","memory"); \
```