

Compiler Examples

- Will use 'gcc' on SUN Sparc to produce some assembly language files for examples
 - ⇒ Do not have a MIPS cross compiler on SUN using GCC
- SPARC is similar to MIPS in that it is a load/store RISC architecture
- Register naming conventions for SPARC
 - ⇒ %o0, %o1, %o2 ... -- registers for passing values to subroutines. These are called \$a0, \$a1,... in MIPS.
 - ⇒ %i0, %i1, %i2, ... -- registers for receiving arguments in subroutine if new Register window has been allocated (o%1 = i%1 in subroutine).
 - ⇒ %fp = framepointer, %sp = stackpointer

Other MIPS vs SPARC differences

- SPARC has register windows (already discussed).
- SPARC has delayed branch
 - ⇒ Instruction after a change of control (branch, jump, subroutine) is always executed
 - ⇒ MIPS has delayed branch also, but it is taken care of in the assembler and the programmer does not see it. The -bare option to XSPIM disables all pseudoinstructions and enables delayed branches, delayed loads.
 - ⇒ Delayed branches are necessary for efficient pipelined implementation (Chapter 6).
- Instructions have SOURCE first, then DEST
 - ⇒ mov %o0, %io ; transfer %o0 to %io

Other MIPS vs SPARC differences

- CALL
 - ⇒ return address placed in OUT7 register (register 15)
 - ⇒ RET instruction does indirect jump using register 15
- JMPL is like CALL except can specify register where return address is placed
 - ⇒ RETL, RET are pseudo instructions that use JMPL to return from a subroutine. Uses register g0.
- No integer multiply instruction

A+B (aplusb.c)

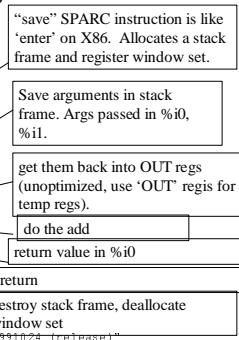
```
aplusb (a,b)
int a,b;
{
    return(a+b);
}

% gcc -S -c aplusb.c
```

No optimization passed to gcc

aplusb.s

```
.file "aplusb.c"
gcc2_compiled:
.section ".text"
.align 4
.global aplusb
.type aplusb,#function
.proc 04
aplusb:
    !#PROLOGUE# 0
    save %sp, -112, %sp
    !#PROLOGUE#
    st %i0, [%fp+68]
    st %i1, [%fp+72]
    ld [%fp+68], %o0
    ld [%fp+72], %o1
    add %o0, %o1, %o0
    mov %o0, %i0
    b .LL2
    nop
.LL2:
    ret
.restore
.LLfel:
    .size aplusb,.LLfel-aplusb
    .ident      "GCC: (GNU) 2.95.2 19991024 (release)"
```



aplusb.s (optimized)

```
% gcc -O -s -c aplusb.c
```

```
.file "aplusb.c"
gcc2_compiled:
.section ".text"
.align 4
.global aplusb
.type aplusb,#function
.proc 04
aplusb:
    !#PROLOGUE# 0
    !#PROLOGUE# 1
    retl
    add %o0, %o1, %o0
.LLfel:
    .size aplusb,.LLfel-aplusb
    .ident      "GCC: (GNU) 2.95.2 19991024 (release)"
```

No stack frame, no window register set.

Just the ADD instruction.

Notice 'add' comes after the return instruction because of delayed branch!!!

M*X + B

```
mxbl {m,x,b)
int m,x,b;
{
    return(m*x+b);
}

% gcc -S -c mxbl.c
```

mxbl.s (optimized)

```
.file    "mxbl.c"
gcc2 _compiled.:
.global .umul
.section ".text"
.align 4
.global mxbl
.type   mxbl, #function
.proc   04
.mprologue
    !#PROLOGUE# 0
    save %sp, -112, %sp
    !#PROLOGUE# 1
    mov  $i0, $o0
    call .umul, 0
    mov  $i1, $o1
    ret
    restore $o0, $i2, $o1
.LLfel:
.size   mxbl,.LLfel-mxbl
.ident  "GCC: [GNU] 2.95.2 19991024 (release)"
```

New window set, sp = sp + -112
No multiply instruction on SPARC. Call a system subroutine for this.
Values (m,x) passed in %i0, %i1.
Return value in %o0
restore window set, also compute final addition of m*x + b. (note that m*x = %o0, %i2 = B.)
