Review: MIPS Register Usage

```
See also: kern/arch/mips/include/asmdefs.h
R0,  zero = ## zero (always returns 0)
R1,  at   = ## (assembler temporary)
R2,  v0   = ## return value / system call number
R3,  v1   = ## return value
R4,  a0   = ## 1st argument (to subroutine)
R5,  a1   = ## 2nd argument
R6,  a2   = ## 3rd argument
R7,  a3   = ## 4th argument
```

Review: MIPS Register Usage

```
R08-R15,  t0-t7 = ## temp variables
R24-R25,  t8-t9 = ## temp variables
                  ## (use without saving)
R16-R23,  s0-s7 = ##  subroutine variables
                  ## (saved/restored by subroutines)
R26-27,   k0-k1 = ## reserved for interrupt handler
R28,      gp    = ## global pointer
                  ## (for easy access to some variables)
R29,      sp    = ## stack pointer
R30,      s8/fp = ## 9th subroutine reg / frame pointer
R31,      ra    = ## return addr (used by jal)
```

How a System Call Works

```
## In OS/161 all user programs are linked with
## lib/crt0/mips-crt0.S which contains __start, the starting
## point for OS/161 programs. See lib/crt0/mips-crt0.S
## for comments on what this code is doing.

004000b0 <__start>:
  4000b0: 3c1c1000  lui gp,0x1000
  4000b4: 279c7ff0  addiu gp,gp,32752
  4000b8: 3c08ffff  lui t0,0xffff
  4000bc: 3508fff8  ori t0,t0,0xfff8
  4000c0: 03a8e824  and sp,sp,t0
  4000c4: 27bdfff0  addiu sp,sp,-16
  4000c8: 3c011000  lui at,0x1000
  4000cc: ac250004  sw  a1,4(at)
```

How a System Call Works

```
4000d0: 0c100040  jal 400100 <main>   # Call main
4000d4: 00000000  nop
4000d8: 00408021  move  s0,v0
4000dc: 0c100050  jal 400140 <exit>
4000e0: 02002021  move  a0,s0
4000e4: 0c100069  jal 4001a4 <_exit>
4000e8: 02002021  move  a0,s0
4000ec: 02002021  move  a0,s0
4000f0: 24020000  li  v0,0
4000f4: 0000000c  syscall
4000f8: 0810003b  j 4000ec <__start+0x3c>
4000fc: 00000000  nop
```

How a System Call Works

```
/* See how a function/system call happens. */
#include <unistd.h>
#include <errno.h>

int
main()
{
  int x;
  int y;
  x = close(999);
  y = errno;

  return x;
}
```

How a System Call Works

```
% cs350-objdump -d syscall > syscall.out
% cat syscall.out

00400100 <main>:
  400100:     27bdffe0     addiu   sp,sp,-32
  400104:     afbf001c     sw      ra,28(sp)
  400108:     afbe0018     sw      s8,24(sp)
  40010c:     03a0f021     move    s8,sp
  400110:     0c10007b     jal     4001ec <close>
  400114:     240403e7     li      a0,999
  400118:     afc20010     sw      v0,16(s8)
  40011c:     3c021000     lui     v0,0x1000
  400120:     8c420000     lw      v0,0(v0)
  400124:     00000000     nop
```

How a System Call Works

```
<main> continued

  400128:     afc20014     sw      v0,20(s8)
  40012c:     8fc20010     lw      v0,16(s8)
  400130:     03c0e821     move    sp,s8
  400134:     8fbf001c     lw      ra,28(sp)
  400138:     8fbe0018     lw      s8,24(sp)
  40013c:     03e00008     jr      ra
  400140:     27bd0020     addiu   sp,sp,32
```

How a System Call Works

```
## See lib/libc/syscalls.S for details/comments */
## At bit easier to understand from disassembled code.

% cs350-objdump -d syscall > syscall.S
% cat syscall.S
...
0040022c <close>:
  40022c: 08100074  j 4001d0 <__syscall>
  400230: 24020007  li  v0,7

00400234 <reboot>:
  400234: 08100074  j 4001d0 <__syscall>
  400238: 24020008  li  v0,8
...
```

How a System Call Works

```
From lib/libc/syscalls.S
   .set noreorder
   .text
   .type __syscall,@function
   .ent __syscall

__syscall:
   syscall          /* make system call */
   beq a3, $0, 1f   /* if a3 is zero, call succeeded */
   nop              /* delay slot */
   sw v0, errno     /* call failed: store errno */
   li v1, -1        /* and force return value to -1 */
   li v0, -1
1:
   j ra             /* return */
   nop              /* delay slot */
   .end __syscall
   .set reorder
```

How a System Call Works

From lib/libc/syscalls.S

```
SYSCALL(close, 7)
SYSCALL(reboot, 8)
SYSCALL(sync, 9)
SYSCALL(sbrk, 10)
SYSCALL(getpid, 11)
```

How a System Call Works

```
syscall instruction generates an exception.
Processor begins execution at virtual address 0x8000 0080
From: kern/arch/mips/mips/exception.S
## Q: where does this address live?  A: kseg0

exception:
  move k1, sp       /* Save previous stack pointer in k1 */
  mfc0 k0, c0_status   /* Get status register */
  andi k0, k0, CST_KUp /* Check the we-were-in-user-mode bit */
  beq  k0, $0, 1f      /* If clear, from kernel, already have stac
  nop                  /* delay slot */

  /* Coming from user mode - load kernel stack into sp */
  la k0, curkstack     /* get address of "curkstack" */
  lw sp, 0(k0)         /* get its value */
  nop                  /* delay slot for the load */

1:
  mfc0 k0, c0_cause    /* Now, load the exception cause. */
  j common_exception   /* Skip to common code */
  nop                  /* delay slot */
```

How a System Call Works

```
From: kern/arch/mips/mips/exception.S
common_exception:
 o saves the contents of the registers
 o calls mips_trap (C code in kern/arch/mips/mips/trap.c)
 o restores the contents of the saved registers
 o rfe (return from exception)

From: kern/arch/mips/mips/trap.c
mips_trap:
  o figures out the exception type/cause
  o calls the appropriate handing function
    (for system call this is mips_syscall).
```

How a System Call Works

From: kern/arch/mips/mips/syscall.c

```
mips_syscall(struct trapframe *tf)
{
  assert(curspl==0);
  callno = tf->tf_v0; retval = 0;

  switch (callno) {
    case SYS_reboot:
      /* is in kern/main/main.c */
      err = sys_reboot(tf->tf_a0);
      break;

    /* Add stuff here */

    default:
      kprintf("Unknown syscall %d\n", callno);
      err = ENOSYS;
      break;
  }
```

How a System Call Works

```
  if (err) {
    tf->tf_v0 = err;
    tf->tf_a3 = 1;      /* signal an error */
  } else {
    /* Success. */
    tf->tf_v0 = retval;
    tf->tf_a3 = 0;      /* signal no error */
  }

   /* Advance the PC, to avoid the syscall again. */
  tf->tf_epc += 4;

  /* Make sure the syscall code didn't forget to lower spl *
  assert(curspl==0);
}
```

C Code for Sections Example

```
#include <unistd.h>

#define N   (200)

int x = 0xdeadbeef;
int y1;
int y2;
int y3;
int array[4096];
char const *str = "Hello World\n";
const int z = 0xabcddcba;

struct example {
  int ypos;
  int xpos;
};
```

C Code for Sections Example (cont'd)

```
int
main()
{
  int count = 0;
  const int value = 1;
  y1 = N;
  y2 = 2;
  count = x + y1;
  y2 = z + y2 + value;

  reboot(RB_POWEROFF);
  return 0; /* avoid compiler warnings */
}
```

cs350-readelf Output: Sections and Segments

```
% cs350-readelf -a segments > readelf.out
% cat readelf.out

Section Headers:
[Nr] Name              Type            Addr     Off    Size   ES Flg  ... Al
[ 0]                   NULL            00000000 000000 000000 00      ...  0
[ 1] .reginfo          MIPS_REGINFO    00400094 000094 000018 18   A  ...  4
[ 2] .text             PROGBITS        004000b0 0000b0 000250 00  AX  ... 16
[ 3] .rodata           PROGBITS        00400300 000300 000020 00   A  ... 16
[ 4] .data             PROGBITS        10000000 001000 000010 00  WA  ... 16
[ 5] .sbss             NOBITS          10000010 001010 000014 00 WAp  ...  4
[ 6] .bss              NOBITS          10000030 00101c 004000 00  WA  ... 16
[ 7] .comment          PROGBITS        00000000 00101c 000036 00      ...  1
...
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor s

Size = 0x250 = 592 bytes
Off  = offset into the ELF file
Addr = virtual address
```

cs350-readelf Output: Sections and Segments

```
Program Headers:
Type     Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
REGINFO  0x000094 0x00400094 0x00400094 0x00018 0x00018 R   0x4
LOAD     0x000000 0x00400000 0x00400000 0x00320 0x00320 R E 0x1000
LOAD     0x001000 0x10000000 0x10000000 0x00010 0x04030 RW  0x1000

 Section to Segment mapping:
  Segment Sections...
   00     .reginfo
   01     .reginfo .text .rodata
   02     .data .sbss .bss

.reginfo = register info (not used)
.text    = code/machine instructions
.rodata  = read-only data (e.g., string literals, consts)
.data    = data (i.e., global variables)
.bss     = Block Started by Symbol
           has a symbol but no value (i.e, uninitialized data)
.sbss    = ?small bss?
```

cs350-objdump -s output: Sections and Segments

```
% cs350-objdump -s segments > objdump.out
% cat objdump.out
...

Contents of section .text:
 4000b0 3c1c1001 279c8000 3c08ffff 3508fff8  <...'...<...5...
...

Decoding 3c1c1001 to determine instruction
0x3c1c1001 = binary 11110000011100000100000000001
0011 1100 0001 1100 0001 0000 0000 0001
001111 | 00000 | 11100 |  0001 0000 0000 0001
instr  | rs    | rt    |   immediate
LUI    | 0     | reg 28|   0x1001
LUI    | unused| reg 28|   0x1001
Load unsigned immediate into rt (register target)
lui gp, 0x1001
```

cs350-objdump -s output: Sections and Segments

```
Contents of section .rodata:
 400300 48656c6c 6f20576f 726c640a 00000000  Hello World.....
 400310 abcddcba 00000000 00000000 00000000  ................
...
## 0x48 = 'H' 0x65 = 'e' 0x0a = '\n' 0x00 = '\0'
## Align next int to 4 byte boundary
## const int z = 0xabcddcba
## If compiler doesn't prevent z from being written/hardware could
## Size = 0x20 = 32 bytes "Hello World\n\0" = 13 + 3 padding = 16
##                +  const int z = 4 = 20
## Then align to the next 16 byte boundry at 32 bytes.
```

cs350-objdump -s output: Sections and Segments

```
Contents of section .data:
 10000000 deadbeef 00400300 00000000 00000000  .....@.........
...
## Size = 0x10 bytes = 16 bytes
## int x = deadbeef (4 bytes)
## char const *str = "Hello World\n"; (4 bytes)
## value stored in str = 0x00400300.
## NOTE: that that is the address of the start
## of the .rodata section (i.e., address of 'H').
```

cs350-objdump -s output: Sections and Segments

```
% cs350-nm -n segments > nm.out
% cat nm.out
...
10000010 A __bss_start
10000010 A _edata
10000010 A _fbss
10000010 S y3
10000014 S y2
10000018 S y1
1000001c S errno
10000020 S __argv
...
## .bss  Size = 0x4000 = 16384 = 4096 * sizeof(int)
10000030 B array
10004030 A _end
```