23 More on MIPS

Outline

- **Announcements**
- **Register Conventions**
- **MIPS Addressing Modes**
- **Basic MIPS Instructions**
- **The SPIM Assembler**

23.1 Announcements

- Turns out there was nothing

23.2 Register Conventions

The MIPS processor has defined certain conventions for register usage as defined in Figure 38. These are conventions (not binding) but failure to honor them will cause other software to fail.

23.3 MIPS Addressing Modes

Format	**Address Computation**
(register)	contents of register
imm	immediate
imm(register)	immediate + contents of register
symbol	address of symbol
symbol ± immediate	address of symbol ± immediate
symbol ± immediate (register)	address of symbol ± (immediate + contents of register)

Note: These are all `imm(reg)`

23.4 Basic MIPS Instructions

Some Basic MIPS instructions are shown in Figure 39. More are listed in the spim manual on the web site as well as the MIPS Overview.

Notice that some of the instructions are pseudo instructions — they are actually synthesized out of "real" instructions. *E.g,* `nop` is really `or 0, 0, 0`

Also notice that instructions come in signed and unsigned flavors. Sometimes they are different — think of division.

23.5 The SPIM Assembler

This is a description of the SPIM assembler. Next week we are going to be doing "Live" code and the assembler syntax will likely be a little different.(Explain why this is and why it doesn't matter.)

Relax. This is all the same as the CUSP assembler, only different.

Segmentation data/text. Default base addresses:

- Data: 0x10010000
- Text: 0x00400000

Entry where to start (*e.g.,* `main`)

Register	Number	Usage
zero	0	Hardwired 0
at	1	Reserved for assembler
v0	2	Function Return Value
v1	3	Function Return Value
a0	4	Argument 1
a1	5	Argument 2
a2	6	Argument 3
a3	7	Argument 4
t0	8	Temporary (Caller Save)
t1	9	Temporary (Caller Save)
t2	10	Temporary (Caller Save)
t3	11	Temporary (Caller Save)
t4	12	Temporary (Caller Save)
t5	13	Temporary (Caller Save)
t6	14	Temporary (Caller Save)
t7	15	Temporary (Caller Save)
s0	16	Saved Temporary (Callee Save)
s1	17	Saved Temporary (Callee Save)
s2	18	Saved Temporary (Callee Save)
s3	19	Saved Temporary (Callee Save)
s4	20	Saved Temporary (Callee Save)
s5	21	Saved Temporary (Callee Save)
s6	22	Saved Temporary (Callee Save)
s7	23	Saved Temporary (Callee Save)
t8	24	Temporary (Caller Save)
t9	25	Temporary (Caller Save)
k0	26	Reserved for OS kernel
k1	27	Reserved for OS kernel
gp	28	Pointer to global area
sp	29	Stack Pointer
fp	30	Frame Pointer (callee save)
ra	31	Return Address (callee save)

Figure 38: The MIPS register conventions

Instruction	**Meaning**
Arithmetic	
`add rd rs rt`	`Addition`
`addi rd rs rt`	`Addition Immediate`
`addu rd rs rt`	`Addition Unsigned`
`addiu rd rs rt`	`Addition Immediate Unsigned`
`(Other arithmetic ops analogous)`	
`and rd rs rt`	`AND`
`andi rd rs rt`	`AND Immediage`
`(Other logical ops analogous)`	
`mult rs rt`	`Multiply (result into HI and LO)`
`multu rs rt`	`Multiply Unsigned`
`(DIV analogous)`	
`mfhi`	`Get the value of HI`
`mflo`	`Get the value of LO`
Control Flow	
`j label`	`Unconditional Jump`
`jr rsch`	`Unconditional Jump via register`
`jal`	`Jump and link (function call)`
`jalr`	`Jump and link via register`
`beq rs rt label`	`Branch if (rs) == (rt)`
`(blt,bgez,blez,` *etc.* `analogous)`	
Load/Store	
`la`	`Load address the computed address`
`lb`	`Load byte (sign extended)`
`lbu`	`Load byte (unsigned)`
`lw`	`Load word (sign extended)`
`lwu`	`Load word (Unsigned)`

Figure 39: Some Basic MIPS Instructions

Assembler Directives Much is the same as the CUSP world.

`.align` n Align on a 2^n boundry. (Why?)

`.ascii, .asciiz` Set up a string

`.byte` Enter Byte Data

`.half` Enter Halfword Data

`.word` Enter Word Data

`.space` n Analogous to .blkw

`.extern` Declare that this exists outside

`.globl` Make this available outside (we'll come back to there two).

The gnu assembler we will use on the CSIF is similar, except it does not recognize symbolic names for registers and insists on numeric register names.

23.5.1 The Virtual Machine

The spim assembler provides a view of a virtual machine which is a little different. It

- provides some pseudoinstructions which is maps to real instructions
- takes care of too-large immediates by generating two instructions
- hides delay slots by filling them with nops