System Hardware

Lecture 4 -Instruction Formats

Objectives

- Understand the factors involved in instruction set architecture design.
- The design of fixed length Instructions.
- Gain familiarity with addressing modes.
- Understanding Instruction Set Architecture (ISA) leads to a deeper understanding of computer architecture.

Introduction

- Instruction formats affect the degree of ease that can be achieved in performing certain operations.
- Simplicity and uniformity are critical to the speed with which a sequence of machine instructions can be ***pipelined*** efficiently.
- The MIPS processor, like many RISC processors, uses fixed-length instructions.
- ***All MIPS instructions are exactly 32 bits***. ***The size of 1 word***.
- Fixed-length instructions offer the advantage of simpler instruction fetching, which also means a smaller, cheaper processor (and possibly faster).

Instructions & Addressing Modes

- A typical machine instruction is **add r1,r2,r3**, which causes the values in **r2** and **r3** to be added, and the sum put in **r1.**
- A machine instruction for an arithmetic/logic operation specifies an opcode, one or more source operands, and, usually, one destination register.
- The opcode is a binary code (bit pattern) that specifies an operation. The operands of an arithmetic or logical instruction can come from a variety of sources.
- The method used to specify where the operands are found, and where the result must go, is called the ***addressing mode*** or ***addressing scheme***. For Simplicity, we will assume that all operands are in registers.

Instruction Formats

The MIPS processor has three different instruction formats:

- Register or R-type instructions
- Immediate or I-type instructions
  - Pure Immediate instructions (I-1)
  - Load, store and branch (I-2)
- Jump or J-type instructions

Instruction Formats

Lets recall a sample MIPS code from Lecture 2

```
loop:   l.d    f0,0(r1)
        add.d  f4,f0,f2
        s.d    f4,0(r1)
        daddiu r1,r1,#-8
        bne r1,r2,loop
        ….
        J done
        ….
done: ….
```

Instruction Formats Register (R-type)

Register or R-type instructions operate on the two registers identified in the 'rs' and 'rt' fields and **Store the result** in register 'rd'.

Examples:

mul.d f4,f2,f6

add r3,r1,r2

opcode	rd (destination)	rs (source 1)	rt (source 2)	Other stuff
6 bits	5 bits	5 bits	5 bits	11 bits
31 26	25 21	20 16	15 11	10 0

Instruction Formats Immediate (I-type)

Immediate or I-type instructions come in two flavors. The general format for an I-type instruction is below. (immediate = operand or offset). The two different types of I-type instructions are presented in the next slides.

opcode	rd (destination)	rs (source)	Immediate
6 bits	5 bits	5 bits	16 bits
31 26	25 21	20 16	15 0

Instruction Formats
I-1 Pure Immediate

I-1 Pure Immediate:

- The 16-bit field in bits 0 - 15 holds a 16-bit integer that plays the same role as '**rt**' in the R-type instructions.
- The specified operation is performed on '**rs**' and the immediate operand (constant), and the result is written into '**rd**', the **destination** register.

Examples: daddiu r1,r1,#-8 <subi r1,r1,8 in RISC-V>

daddiu r1,r1,#8 <addi r1,r1,8 in RISC-V>

Subtracts/adds the 16-bit immediate to '**r1**' to compute a number (mostly a memory address) and write it into '**r1**'.

Instruction Formats
I-2 Instructions

I-2 load, store & conditional branch Instructions:

- The 16-bit field is interpreted as an ***offset***, or ***relative address***, that is added to the ***base*** value in register '**rs**'.
- For ***data access instructions*** (load or store), the offset is the number of ***bytes*** forward (positive) or backward (negative) relative to the base address.
  - **Example: l.d f6,-24(r2)** adds the immediate ***byte-offset*** -24 to '**r2**' to determine a ***memory address***. The double-precision floating point number (64 bits) at that ***memory address*** is then loaded into floating-point register '**f6**'.

Instruction Formats Immediate (I-type)

store Example:

s.d f6, 24(r2) adds the immediate ***byte-offset*** 24 to '**r2**' to determine a ***memory address***. The double-precision floating point number (64 bits) in floating-point register **'f6'** is then stored into that ***memory address.***

Conditional Branch Example:

bne r1,r2,loop compares registers '**r1**' and '**r2**'. If they are not equal, the ***word-offset*** derived from the immediate '**loop**' is ***added to the current value of PC (which is the memory address of the branch instruction)***. That is, we extend the signed integer offset to 18 bits (multiply by 4) to obtain ***word-address*** offset, which is then ***sign-extended*** to 32 bits.

Instruction Formats Immediate (I-type)

branch Instructions:

- For ***branch instructions***, the offset is in ***words***, given that instructions always occupy complete 32-bit memory words.
- To interpret the 16-bit signed integer as a word-address offset, we multiply by 4. Since offsets can be positive or negative, this allows for branching to other instructions within +/- 2^15 (32,768) instructions of the current instruction.

Instruction Formats
Jump Instructions

Jump or J-type Instructions (Unconditional Branch):

opcode	partial jump-target address (word address)
6 bits	26 bits
31 26	25 0

Example: j done, causes ***unconditional transfer*** of control to the instruction at the ***specified word address*** (32 bits).

But only 26 bits are available in the jump address and assumed to specify a word address. ***Two zero bits*** are appended to the right (multiply X4). We now have 28 bits. The four missing bits (on the left) are taken from PC. The ***partial jump-target address*** '**done**' is expanded into a full 32-bit address, and transfer control there.

Addressing Modes Summary

- We have seen that ***addressing mode*** differs from one instruction to another.
- ***addressing mode*** is used within the instruction to specify where operands are found, and where the result must go.
- **Immediate addressing**: An operand is given in the instruction itself. **Example**: daddui r1, r1, #-8 <addi r1,r1,8>
- **Register addressing:** Operands and result are specified in registers. **Example** mul.d f4, f2, f6
- **Base addressing:** An operand is in memory, its location is computed by adding an offset (16-bit signed integer) to the contents of a specified base register. **Example** l.d f6,-24(r2)

Addressing Modes Summary

Addressing modes for branch and jump instructions.

- **PC Relative addressing:** Same as base addressing, except that the "**base**" register is always PC, and a hardware trick is used to extend the constant offset to 18 bits. Namely, we multiply by 4 to obtain a word-address offset, which is then sign extended to 32 bits. This addressing mode is used in conditional branches. **Example:** beq r1, r2, found
- **Absolute addressing**: The addressing mode for unconditional branches is different because there is no "base" register. **Example**: j done '**done**' is a 26-bit natural number. Multiplying by 4 gives a 28-bit natural number. The 4 high order bits of 'done' are then filled with the 4 high order bits of the PC.

Addressing Modes Questions

Q1- Assume 32-bit registers, instructions, and memory addresses, and 16-bit immediates. Consider the instruction **"l.d f6, 7390(r2)".** What is the (8-hexit) hexadecimal representation of the 32-bit integer that will be added to r2.

Q2- Assume 32-bit registers, instructions and memory addresses, and 32-bit immediates. Consider the instruction '**bne r1,r2, loop**', where 'loop' is -2,608. Write the (4-hexit) hexadecimal representation of the low-order 16 bits of the 32-bit integer that will be added to the 'PC'.

Addressing Modes Questions

Q3- Assume 32-bit registers, instructions, and memory addresses, and 16-bit immediates. Consider the instruction **"l.d f6, -31708(r2)".** Write the (8-hexit) hexadecimal representation of the 32-bit integer that will be added to r2.

Q4- Assume 64-bit registers, instructions and memory addresses, and 32-bit immediates. Consider the instruction '**bne r1,r2, loop**', where 'loop' is -20,608. Write the (8-hexit) hexadecimal representation of the low-order 32 bits of the 64-bit integer that will be added to the 'PC'.

Conclusion

- MIPS processor, like many RISC processors, uses 32-bit fixed-length instructions.
- The opcode (control code) is a binary identification code of an instruction (6-bit pattern in MIPS).
- ***addressing mode*** is the method used to specify where operands of an instruction are found, and where the result must go.
- The MIPS processor has 3 different instruction formats (r-type, I-type and J-type) and 5 different addressing modes.