Computer Systems Lecture 4

MIPS Instructions

We will examine the MIPS ISA:

* ISA: Instruction Set Architecture – the language of the computer
* MIPS: a real-world ISA used by many different processors since the 80s (such as the processor used in the PS1).

ISA reference: <http://pages.cs.wisc.edu/~larus/HP_AppA.pdf>

Processor Instructions

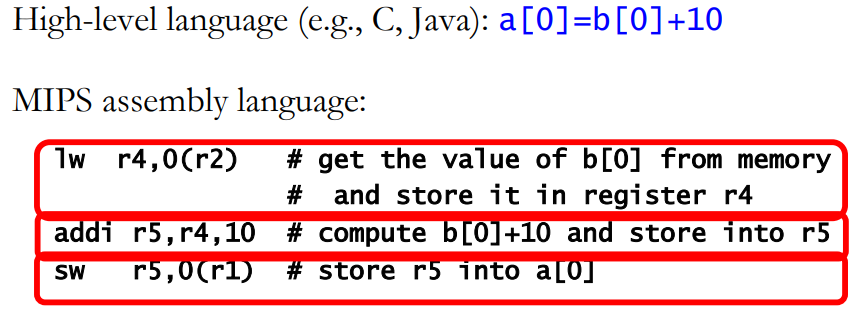
The instruction set architecture is the interface between the software and the hardware. It’s a collection of all the machine instructions recognized by a particular processor. It also deals with privilege levels, memory management etc…

The ISA abstracts away the hardware details from the programmer similarly to how an object hides its implementation detail from tis users, this enables multiple implementation (called microarchitectures) of the same ISA.

Assembly language

Instructions are just strings of binary numbers, for a human it’s very hard to make out which instruction is which. Assembly language is a symbolic representation of machine instructions, it makes it easy for humans to read and write machine code as there is a 1-to-1 mapping between a machine instruction and a corresponding assembly instruction.

MIPS Assembly (Quick Example)



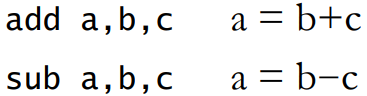
Things to notice here are the separate instructions to access data and to operate on it, MIPS does not allow you to operate on memory directly and that All the instructions have a similar format.

MIPS Arithmetic And Logical Operations

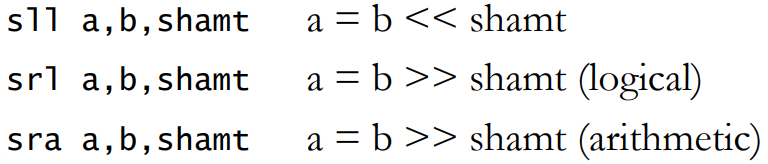
Data processing instructions looks like:



For example:



There are Bit-wise logical instructions: and, or, xor and Shift instructions:



Registers

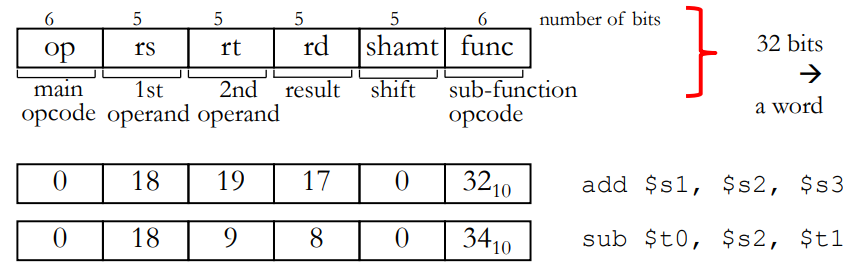
ISA places restrictions on instruction operands, how many there are and where they come from. MIPS processors only operate on registers, registers are storage locations inside the processor that hold program variables and control state. Registers are generally sized to contain the machine’s word (32 or 64 bits). Processors have relatively few programmer-exposed registers, MIPS has 32, x86 has 8-16.

MIPS Instruction Example

Assembly:



Binary (R-format – used for arithmetic instructions):



The shift section of the machine code only applies to shift operations otherwise it’s useless

The main opcode and the sub-function together determine what the operation being done is. (note that result is the destination register).

MIPS Registers

Most registers are available for the programmer to use (with a few important exceptions), Program variables are held in regs $s0-$s7, Temporary variables in $t0-$t9 and the registers with special roles are register 0 ($zero) which is hard wired to the value 0 and the program counter (PC) which holds the address of the next instruction to be executed (this is not a general purpose register), other special registers also exist such as the stack pointer ect…

Immediate Operands

What is we need to operate on a constant like in some arithmetic operations, loop index updates character manipulations etc…?

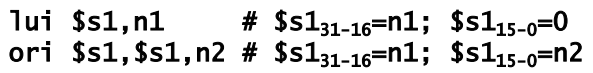
MIPS has dedicated instructions with one constant (immediate) operand e.g. addi $r1, $r1, 1 (r1=r1+1).

Loading a Constant Operand

Constants in MIPS are limited to 16 bits, while the registers are 32, so when we load constants into registers we need to sign extend:



If we have a constant that would require more than 16 bits then we’d need to use an Assembler pseudo-instruction: li reg, constant, which gets translated into two instructions:



Which loads the first 16 bits of the register first (lui) and then the lower 16 bits (ori, which or’s the content of the register with the rest of the constant).