Journal Of Harmonized Research (JOHR)

Journal Of Harmonized Research in Engineering
1(2), 2013, 100-103

ISSN 2347 – 7393

Review Article

INSTRUCTION SET

Shweta Garg, Shrishti Vashist, Shruti Aggarwal

CSE Department, Dronacharya College Of Engineering,
Gurgaon, India

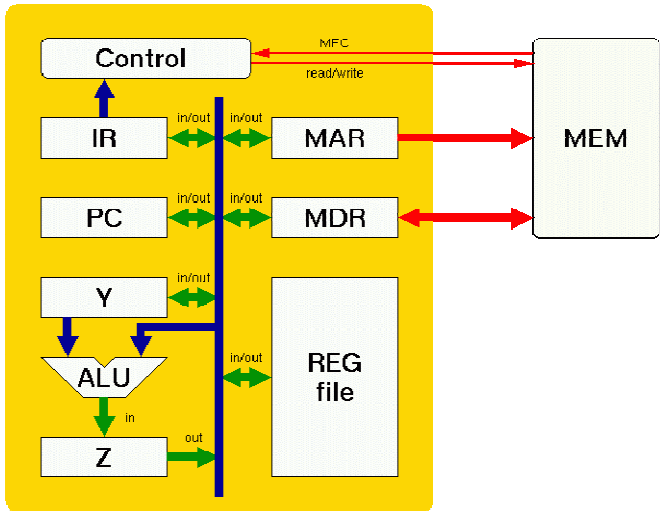
Abstract:
The ideal memory system alleged by most programmers is one which has high capacity, yet allows any word to be accessed immediately. To make the hardware estimated this performance, an increasingly complex memory hierarchy, using caches and techniques like automatic prefetch, has evolved. However, as the gap between processor and memory speeds continues to widen, these programmer-visible mechanisms are becoming inadequate.

Keywords: Explicit- fully and clearly expressed, Retrieved- to recover, Native- being the place, Specifier- to mention,pertaining to.
Abbreviation: ISA-instruction set architecture, I/O-input/output, CISC-complex instruction set computer, RISC-reduced instruction set computer, VLIW-very long instruction word

1. Introduction

An instruction set, or instruction set architecture (ISA), is the part of the computer architecture associated to programming, including the native data types, instructions, registers, addressing modes, memory architecture, interrupt and exception handling, and external I/O. An ISA includes a pattern of the set of opcodes (machine language), and the native commands implemented by a particular processor.

For Correspondence:
shivigarg101ATgmail.com
Received on: October 2013
Accepted after revision: December 2013
Downloaded from: www.johronline.com



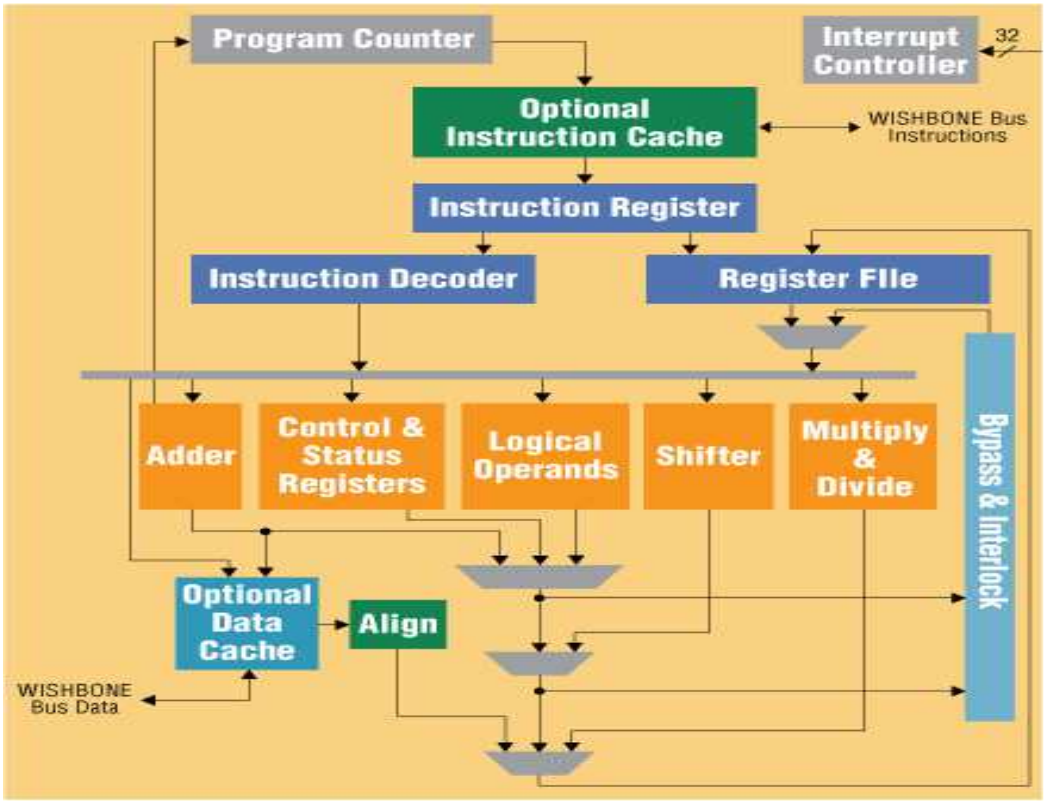
2. Overview

Instruction set architecture is eminent from the microarchitecture, which is the set of processor design techniques used to implement the instruction set. Computers with different microarchitectures can share a common instruction set. For example, the Intel Pentium and the AMD Athlon implement nearly identical versions of the x86 instruction set, but have radically different internal designs.

Some virtual machines that support bytecode, such as Smalltalk the Java virtual Machine, and Microsoft's Common language Runtime virtual machine, as their ISA implement it by translating the bytecode for commonly used code paths into native machine code, and executing less-frequently-used code paths by interpretation; Transmeta implemented the x86 instruction set a top VLIW processors in the same fashion.

3. Classification of instruction sets

A complex instruction set computer (CISC) has many specialized instructions, which may only be rarely used in practical programs. A reduced instruction set computer (RISC) simplifies the processor by only implementing instructions that are frequently used in programs; unusual operations are implemented as subroutines, where the extra processor execution time is offset by their rare use. Theoretically, important types are the minimal instruction set computer and the one instruction set computer, but these are not implemented in commercial processors. Another variation is the very long instruction word (VLIW) where the processor receives many instructions encoded and retrieved in one instruction word.



4. Instruction types

Examples of operations common to many instruction sets include:

4.1. Data handling and memory operations

- **set** a register to a fixed constant value
- **move** data from a memory location to a register, or vice versa. Used to store the contents of a register, result of a computation, or to retrieve stored data to perform a computation on it later.

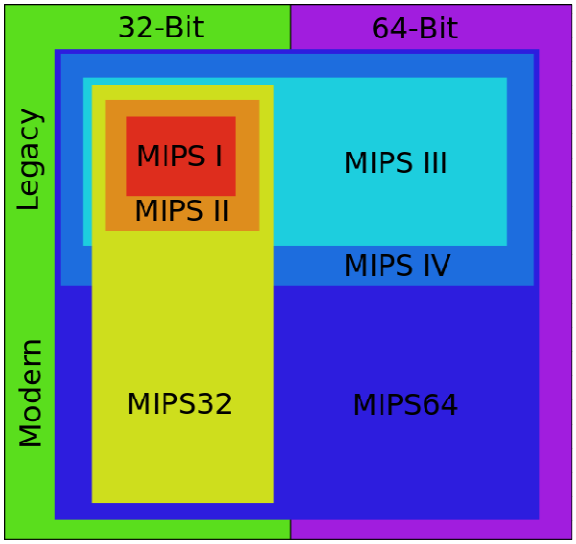
- **read** and **write** data from hardware devices

4.2. Arithmetic and Logic operations

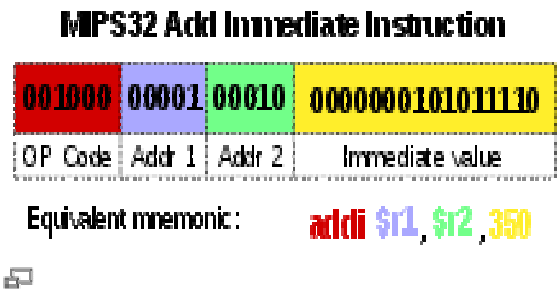
- **add**, **subtract**, **multiply**, or **divide** the values of two registers, placing the result in a register, possibly setting one or more condition codes in a status register
- perform bitwise operations, e.g., taking the **conjunction** and **disjunction** of corresponding bits in a pair of registers, taking the **negation** of each bit in a register
- **compare** two values in registers (for example, to see if one is less, or if they are equal)

4.3. Control flow operations

- **branch** to another location in the program and execute instructions there
- **conditionally branch** to another location if a certain condition holds
- **indirectly branch** to another location, while saving the location of the next instruction as a point to return to



1) 4.4. Parts of an instruction



One instruction may have several fields, which identify the logical operation to be done, and may also include source and destination addresses and constant values. This is the MIPS "Add Immediate" instruction which allows selection of source and destination registers and inclusion of a small constant.

On traditional architectures, an instruction includes an opcode specifying the operation to be performed, such as "add contents of memory to register", and zero or more operand specifiers, which may specify registers, memory locations, or literal data. The operand specifiers may have addressing modes determining their meaning or may be in fixed fields. In very long instruction word(VLIW) architectures, which include many microcode architectures, multiple simultaneous opcodes and operands are specified in a single instruction.

Some exotic instruction sets do not have an opcode field , only operand(s). Other unusual "0-operand" instruction sets lack any operand specifier fields, such as some stack machines.

2) 4.4.1. Number of operands

Instruction sets ma y be categorized by the maximum number of operands *explicitly* specified in instructions.

(In the examples that follow, *a*, *b*, and *c* are (direct or calculated) addresses referring to memory cells, while *reg1* and so on refer to machine registers.)

- 0-operand (*zero-address machines*), so called stack machines: All arithmetic operations take place using the top one or two positions on the stack: **push** *a*, **push** *b*, **add**, **pop** *c*. For stack machines, the terms "0-operand" and "zero-address" apply to arithmetic instructions, but not to all instructions, as 1-operand push and pop instructions are used to access memory.
- 1-operand (*one-address machines*), so called accumulator machines, include early computers and many small microcontrollers: most instructions specify a single right operand (that is, constant, a register, or a memory location), with the implicit accumulator as the left operand

(and the destination if there is one): **load** *a*, **add** *b*, **store** *c*. A related class is practical stack machines which often allow a single explicit operand in arithmetic instructions: **push** *a*, **add** *b*, **pop** *c*.

- 2-operand — many CISC and RISC machines fall under this category:
  - CISC — often **load** *a,reg1*; **add** *reg1,b*; **store** *reg1,c* on machines that are limited to one memory operand per instruction; this may be load and store at the same location
  - CISC — **move** *a->c*; **add** *c+=b*.
  - RISC — Requiring explicit memory loads, the instructions would be: **load** *a,reg1*; **load** *b,reg2*; **add** *reg1,reg2*; **store** *reg2,c*
- 3-operand, allowing better reuse of data:
  - CISC — It becomes either a single instruction: **add** *a,b,c*, or more typically: **move** *a,reg1*; **add** *reg1,b,c* as most machines are limited to two memory operands.
  - RISC — arithmetic instructions use registers only, so explicit 2-operand load/store instructions are needed: **load** *a*, *reg1*; **load** *b,reg2*; **add** *reg1+reg2->reg3*; **store** *reg3,c*; unlike 2-operand or 1-operand, this leaves all three values a, b, and c in registers available for further reuse.
- More operands—some CISC machines permit a variety of addressing modes that allow more than 3 operands (registers or memory accesses), such as the VAX "POLY" polynomial evaluation instruction.

References

1. "Intel® 64 and IA-32 Architectures Software Developer's Manual"
2. Ganssle, Jack Proactive Debugging
3. http://cpushack.net/CPU/cpu7.html
4. The evolution of RISC technology at IBM by John Cocke