Instruction Set Architecture

Instruction Set Architecture 1

CSE2CSA Lecture 13

Instruction Set Architecture - Classification

Classifying Instruction Set Architectures

Instruction set architecture (ISA) is that portion of the machine visible to the programmer or compiler writer. We will mainly focus on:-

- Taxonomy of Instruction Set alternatives and some qualitative assessment (advantages/disadvantages) of each approach.
- Analyse some instruction set measurements that are largely independent of a specific instruction set.
- Instruction Sets not aimed at general (desktop) computers (brief).
- Issues of language and compilers and their effect on instruction set architecture.
- Briefly consider the MIPS and Intel (80x86) instruction sets (Intel uses RISC internally while supporting 80x86 externally).

Instruction Set Architecture - Classification

Instruction Set

The instruction set is a set of instructions that a processor understands and can execute – in effect it defines the processor. The set of instructions (⇔ instruction set) for each type of processor is "*hardwired into*" the processor and may be completely different from one machine/CPU type to another. Typically, different instruction sets differ in the sizes of instructions, kind of operations they perform or allow, type of operands they operate on, and the type of results they provide. Each of these parameters may have a dramatic affect on the performance of any one type of machine.

Computer systems are often defined by the type of CPU that is incorporated into the system. For example programs compiled on an IBM PC (or compatible) system use the instruction set of an 80x86 CPU. All high level languages are compiled/interpreted into these instructions before a CPU can execute the program. {The exception is (Java) byte-code which is machine language for a **virtual machine**.}

Each instruction is normally written as a type of assembler language and is considered, in this course, as a 1 to 1 representation of the machine code that the CPU understands. (⇔ one line of assembler code = one instruction = one machine instruction). We will use a general "brief" instruction format when considering instruction sets in this course. (See Later).

Classification by Opcode

- Each instruction consists of an opcode (saying what sort of instruction it is) followed by 1, 2 or 3 operands which are the data input & output values of the instruction.
- Instruction sets can be classified by the number and types of opcodes or by how inputs and outputs are specified in the operands.
- Classifications by opcode are:
  - Arithmetic operations eg Add a b c
  - Data transfer instructions eg Load a or Store a
  - Decision making instructions eg IfNonZero a
  - Jump instructions eg Jump a

Instruction Set Architecture - Classification

The type of intended storage in the CPU is the most basic differentiation – the major choices are :-

- Stack
- Accumulator
- Register (set)

Typically, a set of axes for alternative design choices in instruction sets is given below.

Operand storage in the CPU	Where are operands kept other than in memory?
Number of explicit operands named per instruction	How many operands are named explicitly in a typical instruction?
Operand location	Can any ALU instruction operand be located in memory or must some or all of the operands be internal storage in the CPU? If an operand is located in memory, how is the memory location specified?
Operations	What operations are provided in the instruction set?
Type and size of operands	What is the type and size of each operand and how is it specified?

Instruction Set Architecture - Classification

In various architectures operands may be named explicitly or implicitly. For example operands in a stack architecture are implicitly on top of the stack, while in an accumulator architecture one operand is implicitly the accumulator, while general-purpose-register architectures (GPR) only have explicit operands. The following table summarizes these differences.

Temporary storage provided	Examples	Explicit operands per ALU	Destination for results	Procedure for accessing explicit operands
Stack	B5500 HP3000/70	0 (all implicit)	Stack	Push and Pop onto or from stack
Accumulator	PDP-8, Motorola 6809	1 (and 2nd is implicit)	Accumulator	Load/store of Accumulator
Register set	IBM 360, DEC VAX	2 or 3 (all explicit)	Register or memory	Load/store of register or memory access

Instruction Set Architecture - Classification

There are two main classes of register set machines: -

- Access memory as part of any instruction – called **register-memory architecture,**
- Access memory only with load and store instructions – called **load-store** or **register-register architecture**.

There is a third class called a **memory-memory architecture** where all operands are directly accessed from memory (no direct usage of registers) however these are not generally used in general purpose machines today. The reason for this is that registers are faster then memory, easier for a compiler to use and, more importantly, registers can be used to hold often accessed variables, which, when allocated to registers can significantly reduce memory traffic, allowing the program to run faster. This is due to:-

- Registers are faster than memory accesses, and
- Code density increases (Registers can be named with fewer bits than a memory location)

Instruction Set Architecture - Classification

To demonstrate the differences between each type of instruction set, consider the following code sequence:-

int A, B, C; /* 3 typical variables declared as int's. */

C = A + B; /* Add A to B and store the result in C*/

For the following 4 instruction set types we assume that A, B and C all belong in memory and the original values of A and B are to be maintained:-

Stack	Accumulator	Register -memory	Register-register (⇔Load-Store)
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R3,R1,B	Load R2, B
Add	Store C	Store R3,C	Add R3, R1, R2
Pop C			Store R3, C

Note that we are using general description instruction mnemonics here (push, store, pop … not a specific instructions such as what is found on an Intel or similar chipset)

- R1, R2, etc. are the specific General Purpose registers in the CPU,
- A, B & C represent general memory addresses.

Instruction Set Architecture - Classification

Note that we will use the following general terminology when considering instruction sets and their various formats:-

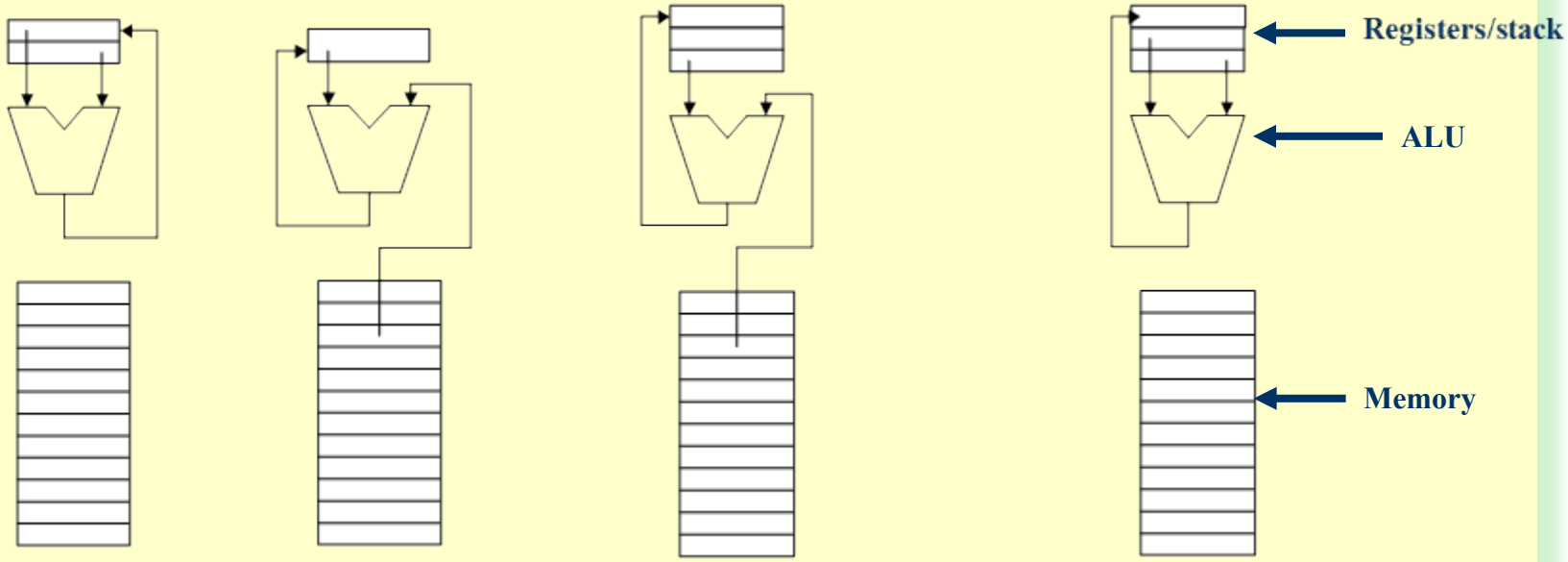
- **Instruction** (also **called op field or opcode**) ⇔ e.g. Load …, Add…, Mult…, Store…, etc. represents the actual instruction, while the "…" represents operands associated with each instruction which vary according to the architecture class and operand set formats of the machine.

- **Operand** ⇔ the sources of the data being operated on, e.g.
  - Load R1, A; ⇔ Has two operands (R1 and A); where R1 refers to a specific register (⇔R1) and A refers to an address in memory.
  - Add R3, R2, R1, ⇔ Has 3 operands (R3, R2, R1) and refer to 3 different registers.
  - Store R3, C, ⇔ Has 2 operands (R3, C); one is a register (R3), while the other is a memory address (C).

Instruction Set Architecture - Classification

- The accompanying diagram outlines the operand locations for the 4 possible Instruction set classes with the arrow head indicating the destination and the base of the arrow staff indicating the source:-

(a) Stack **(b) Accumulator** **(c) Register-memory** **(d) Register – register (Load-store)**



Instruction Set Architecture - Classification

The primary advantages/disadvantages of stack, accumulator and register are:-

Machine type	Advantage	Disadvantage
Stack	Simple mode of expression evaluation (reverse polish). Short instructions can yield good code density.	A stack cannot be randomly accessed. This limitation makes it difficult generate efficient code. It's also difficult to implement efficiently as the stack becomes a bottle neck.
Accumulator	Minimizes internal state of machine. Short instructions.	Since accumulator is only temporary storage, memory traffic is higher for this approach.
Register	Most general mode for code generation	All operands must be named, leading to longer instructions.

However the *register* classification (above) can be further divided into 2 main classifications:-

1. Register-memory, and
2. Register-register (Load-store)

Additionally, we have a further categorization called General Purpose Register Machines (GPR) that has evolved with time and are largely represented by the popular machines today.

Instruction Set Architecture - Classification

Classifying General-Purpose Register (GPR) Machines (Operand storage in memory).

There are two major instruction set characteristics which divide GPR architectures:

1. Whether an ALU instruction has 2 or 3 operands, and
2. How many of the operands in an ALU operation may be memory addresses.

(these will be defined shortly)

The key advantages of general purpose register machines arise from effective use of the registers by a compiler in computing expression values and in using registers to hold variables.

Registers also permit more flexible ordering in evaluating expressions than stacks or accumulators. e.g.