

## Instruction Sets

- An Instruction Set provides a *functional* description of a processor.
  - a detailed list of the instructions that the processor is capable of processing.
  - a description of the types/locations/access methods for *operands*.

---

---

---

---

---

---

---

---

## Instruction Set Architecture

- Most instruction sets are very similar.
- There have been a few different ways to define the location of operands.
  - operands are the data that are operated on, for example the numbers added together in an addition instruction.
- There is one primary architecture in use now, but it's worth looking at the others.

---

---

---

---

---

---

---

---

## Stack Architecture

- In a stack-based instruction set the processor supports the notion of a *stack*:
  - A stack is a *last-in-first-out* list.
  - putting something on the stack is called a *push*
  - getting something off the stack is a *pop*
- In a stack-based instruction set all operands are on *the stack*.

---

---

---

---

---

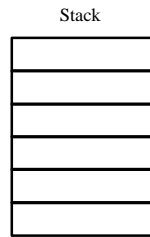
---

---

---

## Stack Example: $A=B+C$

Program  
**push B**  
**push C**  
**add**  
**pop A**



CompOrg Instruction Set Architecture

4

---

---

---

---

---

---

---

---

## Stack Architecture: Implicit Operands

- In the Add instruction the operands are not explicitly defined in the instruction.
  - There are no bits in the machine code that are used to encode the location of the operands.
- The location where the result should be placed is also implicit (the top of the stack).

CompOrg Instruction Set Architecture

5

---

---

---

---

---

---

---

---

## Stack Architectures: Pros & Cons

- Small instructions (don't need many bits to specify the operation).
- Few options (compiler writer has it easy).
- Lots of memory accesses required
  - everything that is not on the stack is in memory.

CompOrg Instruction Set Architecture

6

---

---

---

---

---

---

---

---

## Notes on Stack Architectures

- The stack instruction set architecture has not been used for many years...
- People are talking about a hardware implementation of the Java *virtual machine* that is stack based!

CompOrg Instruction Set Architecture

7

---

---

---

---

---

---

---

## Accumulator Instruction Set Architecture

- Many early processors were based on a different way to support *implicit* operands.
- A single *word* of internal memory called the *accumulator* is always one of the operands.
- The result of an operation is always stored in the *accumulator*.

CompOrg Instruction Set Architecture

8

---

---

---

---

---

---

---

## Accumulator Example: $A=B+C$

<b>Load</b>	<b>B</b>	<b># Acc = B</b>
<b>Add</b>	<b>C</b>	<b># Acc = Acc + C</b>
<b>Store</b>	<b>A</b>	<b># A = Acc</b>

CompOrg Instruction Set Architecture

9

---

---

---

---

---

---

---

## Accumulator Pros and Cons

- Easier to implement than stack.
- Small instructions (one implicit operand)
- More memory access required than stack.
  - Lots of *spill code* necessary. Consider a program to do this:

$$A=B*C+D*E$$

CompOrg Instruction Set Architecture

10

---

---

---

---

---

---

---

---

## General Purpose Register Architecture

- A Register is a *word* of internal memory (like the accumulator).
- A General Purpose Register architecture supports many registers – each can be used for anything:
  - holding operands for operations
  - holding temporary values

CompOrg Instruction Set Architecture

11

---

---

---

---

---

---

---

---

## Number of Registers

- Early Register Instruction Sets supported a few registers (8 or less).
- Many current processors support 32 registers.
- The more registers available, the fewer memory accesses will be necessary.

CompOrg Instruction Set Architecture

12

---

---

---

---

---

---

---

---

## Register Example: A=B+C

```
Load R1,B      # R1 = B
Load R2,C      # R2 = C
Add  R3,R1,R2  # R3 = R1+R2
Store R3,A     # A = R3
```

CompOrg Instruction Set Architecture

13

---

---

---

---

---

---

---

---

## Register Pros and Cons

- Instructions must include bits to specify which registers to operate on (large instruction size).
- Memory access can be minimized (registers can hold lots of intermediate values).
- Compiler writer now has to attempt to maximize register usage (minimize *spill code*). This is a tough job!

CompOrg Instruction Set Architecture

14

---

---

---

---

---

---

---

---

## Modern Processors

- Most processors in use today are register based.
  - there are still a number of microcontrollers that are widely used that are accumulator based.
- Pentium, MIPS, Sparc, Alpha, PowerPC, PA-RISC are all G.P. Register architectures.

CompOrg Instruction Set Architecture

15

---

---

---

---

---

---

---

---