Instruction Set Architectures

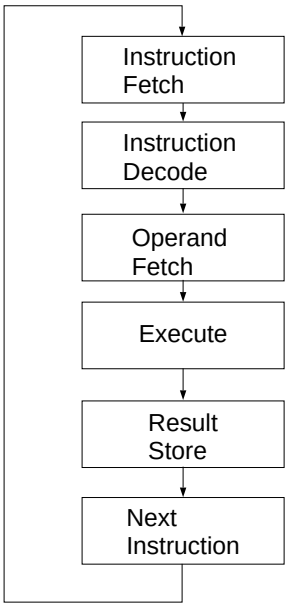
Interface between hardware and low-level software

Goal: Find a language that makes it easy to build both the hardware and the compiler while maximizing performance and minimizing cost

Programmer's View - add, subtract, and, or, compare, ...

Computer's View - strings of 1s and 0s

Execution Cycle



Basic Issues In Instruction Set Design

What operations (and how many) should be provided

How (and how many) operands are specified

What data types and sizes

How to encode these into consistent instruction formats

Typical Operations

Data Movement	load/store (from/to memory)
	memory-to-memory move
	register-to-register move
	input/output (from/to I/O device)
	push/pop (to/from stack)
Arithmetic	integer (binary + decimal) or FP
	add, subtract, multiply, divide
Logical	not, and, or, set, clear
Shift	shift left/right, rotate left/right
Control (Jump/Branch)	unconditional, conditional
Subroutine Linkage	call, return
Interrupt	trap, return
Synchronization	test&set (atomic read-modify-write)
String	search, translate

Control Flow

Need to be able to test conditions for conditional control transfers

Four basic conditions: N – negative; Z – zero; V – overflow; C – carry;

Sixteen combinations of the basic four conditions:

Always	Unconditional
Never	NOP
Not Equal	$\sim Z$
Equal	Z
Greater	$\sim [Z + (N \bigoplus V)]$
Less or Equal	$Z + (N \bigoplus V)$
Greater or Equal	$\sim (N \bigoplus V)$
Less	$N \bigoplus V$
Greater Unsigned	$\sim (C + Z)$
Less or Equal Unsigned	$C + Z$
Carry Clear	$\sim C$
Carry Set	C
Positive	$\sim N$
Negative	N
Overflow Clear	$\sim V$
Overflow Set	V

Methods of Testing Conditions

- Condition Codes: Processor status bits are set as a side effect of arithmetic instructions (possibly on moves), or explicitly by compare or test instructions.

```
ex:  add r1, r2, r3
     bz label
```

- Condition Register:

```
ex:  cmp r1, r2, r3
     bgt r1, label
```

- Compare and Branch: combine the above two instructions

```
ex:  bgt r1, r2, label
```

Addressing Modes

Addressing mode	Example	Meaning
Register	Add R4,R3	R4←R4+R3
Immediate	Add R4,#3	R4←R4+3
Displacement	Add R4,100(R1)	R4←R4+Mem[100+R1]
Register indirect	Add R4,(R1)	R4←R4+Mem[R1]
Indexed	Add R3,(R1+R2)	R3←R3+Mem[R1+R2]
Direct or absolute	Add R1,(1001)	R1←R1+Mem[1001]
Memory indirect	Add R1,@(R3)	R1←R1+Mem[Mem[R3]]
Auto-increment	Add R1,(R2)+	R1←R1+Mem[R2]; R2←R2+d
Auto-decrement	Add R1,-(R2)	R2←R2-d; R1←R1+Mem[R2]
Scaled	Add R1,100(R2)[R3]	R1←R1+Mem[100+R2+R3*d

Instruction Format

Can be fixed length, variable length, or a hybrid

If we have many memory operands per instruction and many addressing modes, we need an Address Specifier per node

If we have a load-store machine with 1 address per instruction and one or two addressing modes, then we can encode the addressing mode in the opcode

Data Types and Sizes

- Bit: 0, 1
- Bit String: sequence of bits of a particulat length — 4 bits=nibble, 8 bits=byte, 16 bits=half-word, 32 bits=word
- Character: ASCII - 7 bit code, EBCDIC - 8 bit code
- Decimal: digits 0-9 encoded as 0000b through 1001b, two decimal digits packed per 8 bit byte
- Integers: sign&magnitude, 1's complement, or 2's complement representation
- Floating Point: single precision, double precision, extended precision - $M \times R^E$, where M=mantissa, R=base, E=exponent