Evolution of ISAs

Characterizing ISAs

- A traditional way to look at ISA complexity encompasses:
- *Number of operands per instruction*:
  - How many operands are specified in an instruction?
  - Is the number of operands fixed?
- *Number of addresses per instruction*: How many of the operands can be memory addresses?
- *Regularity of instruction formats*:
  - Variable length or fixed length?
  - Few or many formats?
- *Number of addressing modes/types*.
- *Registers*
  - Special purpose (reserved) or general purpose?
  - Are the registers implied/specified in the instruction?

A Tour of Common Addressing Modes

Name	**Example**	**Meaning**
* Immediate	100	100
* Register	r6	Contents of r6
Register Deferred	(r6)	Memory[r6]
* Based/Displacement	100(r6)	Memory[r6+100]
* PC-Relative	100	PC + 100
Deferred	@100(r6)	Memory[Memory[r6+100]]
Autoincrement	(r3)+	Memory[r3]; r3 = r3 + size
Autodecrement	-(r3)	r3 = r3 - size; Memory[r3]
Autoincrement deferred	@(r3)+	Memory[Memory[r3]]; r3 = r3 + size

- We use VAX-like asm notation for non-MIPS addr modes...

Accumulator Machines

- Early machines, and many microcontroller (Motorola 6809) used an implied register called an accumulator.
  - Operands per instruction: at most 1
  - Addresses per instruction: at most 1
  - Instruction formats: fixed length, few formats for ease of programming
  - Addressing modes: few (typically immediate and and PC-relative)
  - Registers: one implied register
- How would we encode A = B + C; ?

```
load    addressB
add     addressC
store   addressA
```


Stack Machines

- Machines where all data is on an implied stack
  - Operands per instruction: at most 1
  - Addresses per instruction: at most 1
  - Instruction formats: variable length, few formats
  - Addressing modes: few (typically immediate and and PC-relative)
  - Registers: none (for performance, there are often “hidden” registers)
- How would we encode A = B + C; ?

```
push    addressB
push    addressC
add
pop     addressC
```


CISC Machines

- Intel 80x86, Motorola 680x0 are examples of CISC machines.
- They are *register-memory* architectures (a few operands are allowed to be memory addresses)
  - Operands per instruction: variable, up to 2
  - Addresses per instruction: 1
  - Instruction formats: variable length (80x86 instructions are between 1 and 6 bytes), many formats
  - Addressing modes: 80x86 has at least 7, 68k has more
  - Registers: usually a few special purpose and a few general purpose (80x86 has 8 special purpose, 8 fp; 680x0 has 8 data, 8 address)
- How would we encode A = B + C; ?

```
load    r1, addressB
add     r1, addressC
store   r1, addressA
```


True CISC

- The VAX is the ultimate CISC.
  - Operands per instruction: variable, up to 3
  - Addresses per instruction: variable, up to 3
  - Instruction formats: variable length (1 to 54 bytes), many formats
  - Addressing modes: more than 10
  - Registers: 16 general purpose
- How would we encode A = B + C; ?

```
add     addressA, addressB, addressC
```

- VAX also included special loop instructions, as well as call and return instructions
- Compared to the one MIPS add instruction, VAX has many versions, depending on # of operands and addressing modes, leading to thousands of different combinations.
- This complicates the implementation of the processor tremendously.

RISCs

- Typically *load-store* or *register-register* architectures
  - Operands per instruction: 3
  - Addresses per instruction: 0 (must use load/store operations to move data between memory and registers)
  - Instruction formats: fixed length, few formats
  - Addressing modes: few (MIPS has immediate, register, based, and PC-relative)
  - Registers: many general purpose
- How would we encode A = B + C; ?

```
lw      r1, offsetA(r5)
lw      r2, offsetB(r5)
add     r3, r2, r1
sw      r3, offsetC(r5)
```


Summary Comparisons

	Accumulator	Stack	CISC	RISC
Implementation	easy	easy	hard	easy
Instruction density	high	high	high	low
Assembly coding	easy	medium	easiest	tiresome
Compilation	easy	easy	easy	hard
Memory overhead	high	high	highest?	lower
Instruction count	medium	medium	low	high
CPI	?	?	high	low
Cycle time	?	?	high	low

- If RISCs have high instruction count, how can they possibly achieve such good performance?

Historical Trends

- The 60s: expensive memory, poor compiler technologies, poor implementation technologies.
  - Goals: simple compilers (or assembling), simple hardware implementation, high code density
  - Results: simple ISA, regular formats, compact encoding
- The 70s: advances in implementation technologies, poor compilers, expensive memory, high software costs
  - Goals: simple compilers, high code density
  - Results: powerful ISA, irregular formats, compact encoding, complicated implementations
- The 80s saw advances in implementation tech, advanced compilers, cheap memory
  - Goals: high performance by pipelining, simple implementation, compat. w/ optimizing compilers
  - Results: simple ISA, regular formats, lots of registers

Some Modern Processors

Processor	Mhz	Year	Style	Transistors x 10^6	SpecInt/Fp92
Motorola 68040	25	1989	CISC	1.2	21/15
Intel 80486DX	50	1991	CISC	1.2	33/15
R4000	100	1992	RISC	1.1	59/60
PPC 620	133	1995	RISC	7.0	225/300
Dec Alpha 21164	300	1995	RISC	9.3	330/500
Intel P6	166	1996	CISC	5.5	290/260
R10000	200	1996	RISC	6.8	300/600
Sun Sparc Ultra II	250	1996	RISC	5.4	350/550
HP 8000	200	1996	RISC	3.9	360+/550+