03-1

Addressing Modes and Instruction Set Design

To be covered:

- Architecturally visible storage.
- ISA classification based on register organization.
- Addressing modes.
- Impact on ISA.

Architecturally Visible Storage

Two types used: *registers* and *memory*.

Registers (Internal Storage)

Store what is actively being worked on.
E.g. Math expression parts, array indices.

Implemented using highest speed memory.

Given short names.
E.g. `r1, g1, AL`.

Small number of registers provided.
E.g. 32, 64.

Goal: fastest access.

Memory

Stores code and data.

Simple to programmer ... despite complex implementation.

Large number of locations, 2^{32} and 2^{64} are common.

Named using integers called *addresses*...
...and some address space identifier.

Goal: large size.

03-3

ISA Classification

Classified by Register Organization

Listed below by distinguishing features...
...and *intended* (contemporary) benefits.

Next slide: current relevance.

Load/Store, General-Purpose Registers (GPR)

Memory $\leftrightarrow$ register movement uses *Load* and *Store* instructions.

ALU instructions refer to registers, not memory.

Number of special-purpose registers minimized.

$\Rightarrow$Keep memory and ALU operations separate.

$\Rightarrow$Avoid special-purpose instructions.

Memory/Memory, General-Purpose Registers

Load and store instructions, and also...

...ALU instructions refer to registers *and* memory.

Few special-purpose registers.

$\Rightarrow$Use registers only when needed.

Accumulator

Typical ALU instruction uses...
...a special *accumulator* register...
...and a general purpose register.

⇒Keep instructions small.

Stack

Instructions to push and pop data from stack.

ALU instructions refer to stack.

⇒Natural way to evaluate expressions.

Tradeoffs of ISA Types

Load/Store, General-Purpose Registers (GPR)

Used in most ISAs developed in past decade.

+ Programmer- and compiler-friendly.
  (Since most registers can be used for any purpose.)

+ Allow single-size instruction coding.
  (Since multiple memory addresses not needed in ALU instr.)

Memory/Memory, General-Purpose Registers

+ Lots of addressing options for programmers.

– Lots of addressing options forces slower implementation.

Accumulator

– Extra instructions needed to move data.

Stack

+ Programs small.

– Implementations slow.

– Hard to code certain expressions.

Internal Storage Variations

Operands per Instruction

Three typically used.

Two sometimes used.

Factors:

Instruction coding (bits to specify operands).

Addresses per ALU Instruction

Zero typically used (load/store).

One, two, even three sometimes.

Factors

Instruction coding.
(Addresses take up lots of space.)

Benefit over multiple instructions.

03-7

Memory Addressing

Address Interpretation

Sequence of memory locations (usually bytes) starting at address.

Size of sequence depends upon instruction.

E.g., `LW`, load word instruction might read four bytes.

E.g., `LB`, load byte instruction might read one byte.

Alignment

Addresses subject to *alignment* restrictions...
...when used in certain instructions.

E.g., a *word-aligned* address must be divisible by 4 (usual word size).

03-8

Addressing Modes, Overview

Many instructions specify data for operations.

Where data (operand) might be:

Part of instruction.

Called an *immediate* value.

Register.

Instruction specifies which register.

Memory

Instruction contains address of data...

...or instruction specifies register containing address of data...

...or instruction specifies how address computed...
...using registers, constants, and values retrieved from memory.

Common Addressing Modes

Register

Data in register.

```
Move R4, R3 !  R4 = R3 (This is a comment.)
```

Immediate

Data in instruction.

```
Move R4, #3 !  R4 = 3
```

Register Deferred or *Register Indirect*

Data address in register.

```
Load R4, (R1) !  R4 = MEM[R1]
```

Displacement

Data address is register plus constant.

```
Load R4, 100(R1) !  R4 = MEM[ R1 + 100 ]
```

Indexed

Data address is sum of two registers.

```
Load R4, (R1+R2) !  R4 = MEM[ R1 + R2 ]
```

Direct

Data address is a constant.

```
Load R1, (1024) !  R1 = MEM[ 1024 ]
```

Memory Indirect

Address of data's address is in register.

```
Load R1,@(R3) !  R1 = MEM[ MEM[ R3 ] ].
```

Autoincrement

Perform register indirect access, then add constant to register.

```
Load R1,(R2)+ !  R1 = MEM[ R2 ]; R2 = R2 + 1
```

Autodecrement

Subtract constant from register then perform register indirect access.

```
Load R1,-(R2) | R2 = R2 - 1; R1 = MEM[ R2 ];
```

Scaled

Data address is constant1 + reg1 + reg2 * constant2.

```
Load R1,100(R2)[R3] !  R1 = MEM[ 100 + R2 + R3 × d ]
```

There's no limit to how many addressing modes one could think of.

03-11

Memory Addressing Choices in ISA Design

Which addressing modes?

Affects cost and may limit future performance.

Which instructions get which addressing modes?

Affects cost and may limit future performance.

Maximum displacement size?

Limited by instruction size.

Maximum immediate size?

Limited by instruction size.

03-12

Past Choices

Vax (Digital Equipment Corporation)

Older ISA, variable instruction size.

Provides many addressing modes usable by many instructions.

Instructions use 32-bit immediate.

\+ (ISA) Compact code.

– (Imp) Future implementations limited.
So limited that DEC developed a new ISA, Alpha.

– (Imp) Expensive-to-implement modes go unused.

SPARC V8 (Sun Microsystems)

Newer ISA, fixed 32-bit instruction size.

Load/Store instruction modes: indexed and 13-bit displacement.

ALU instruction modes: register and 13-bit immediate.

`sethi` instruction modes: 22-bit immediate.
(Used for moving large immediate into registers.)

\+ (Imp) Uniform instruction sizes.

\+ (Imp) Useful modes included.