# CS4617 Computer Architecture

## Lecture 7: Instruction Set Architectures

Dr J Vaughan

October 1, 2014

## ISA Classification

- Stack architecture: operands on top of stack
- Accumulator architecture: 1 operand in ACC, implicitly
- General-purpose register architectures: Explicit operands in register or memory

## Register computers: 2 main classes

- Register-memory architecture: Access memory as part of any instruction
- Load-store architecture: Access memory only with load and store
- Other: Extended accumulator/special-purpose register: use additional registers in special ways

## Advantages of general-purpose register (GPR) computers

- Registers are faster than memory
- Registers are more efficient for a compiler to use
- Registers allow expression evaluation in any order

## 2 design decisions for GPR architectures

1. Does ALU instruction have 2 or 3 operands?
2. How many ALU operands may be memory addresses?

## Expectations for a new ISA design

- General-purpose registers
- Load-store version of GPR

## Memory Addressing

- Endiannness
  - Little endian: Low-order bytes placed in lower addresses
  - Big endian: Low-order bytes placed in higher addresses
- Alignment
  - Access to object of $s$ bytes is aligned if $A\ mod\ s = 0$

## Addressing Modes

- Register
- Immediate
- Displacement
- Register indirect
- Indexed
- Direct (also known as Absolute)
- Memory Indirect
- Autoincrement
- Autodecrement
- Scaled

## Expectations for ISA addressing modes

1. Displacement, immediate and register indirect are used in 75% to 95% of instructions
2. Size of displacement field at least 12 to 16 bits
3. Size of immediate field at least 8 to 16 bits

## Operand Types

- Type gives size
- Character 8 bits
- Half word 16 bits
- Word 32 bits
- Single-precision floating point word 32 bits
- Double-precision floating point word 64 bits

## Operand Representation

- Integer: Two's complement binary
- Character: 8-bit ASCII, 16-bit Unicode
- Floating-point: IEEE 754
- Decimal numbers: Packed/unpacked BCD

## Operations at ISA level

- Arithmetic and logical
- Data transfer
- Control
- System: SVC, memory management
- Floating point
- Decimal
- String: Move, compare, search
- Graphics: Pixel and vortex operations, compress/decompress

## Top ten 80x86 instructions

| **Instruction** | **Usage frequency** |
|---|---|
| Load | 22% |
| Conditional branch | 20% |
| Compare | 16% |
| Store | 12% |
| Add | 8% |
| And | 6% |
| Sub | 5% |
| Move reg-reg | 4% |
| Call | 1% |
| Return | 1% |
| **Total** | 96% |

Table: Branch condition testing

## Control Flow Instructions

| **Instruction type** | **Usage frequency** |
|---|---|
| Conditional branches | 75% |
| Jumps | 6% |
| Procedure calls | 19% |
| Procedure returns | |

Table: Control flow instruction frequency in integer benchmarks

## Control flow addressing modes

- Most frequent branches in integer programs are to targets that can be encoded in 4 to 8 bits.
- Displacement to be added to PC
- PC-relative branches/jumps
- Advantage
  - Code runs independently of load address
  - Less work for linker

## Returns and indirect jumps

- PC-relative cannot be used for returns/indirect jumps
- Must be able to specify destination address dynamically so that it can change at run time
- Possibility: name register containing target address
- Possibility: allow any addressing mode for jump target specification

## Uses of register indirect jumps

- Case/switch statements
- Virtual functions/methods in OO languages
- Function pointers that allow functions to be passed as parameters
- Dynamically shared libraries, loaded and linked at runtime
- In these four cases, the target address is not known at compile time

## Branch condition testing

| **Method** | **Examples** | **How tested** |
|---|---|---|
| Condition Code | 80x86, ARM, Power PC | Test special flag bits |
| Condition register | Alpha, MIPS | |
| Compare and branch | PA-RISC, VAX | Compare is part of the branch |

Table: Branch condition testing

## Comparisons

- Many comparisons are simple tests
- A large number of comparisons are with zero

| **Comparison** | **Usage frequency** |
|---|---|
| Not equal | 2% |
| Equal | 18% |
| Greater than or equal | 11% |
| Greater than | 0% |
| Less than or equal | 33% |
| Less than | 35% |

Table: Comparison frequency in integer benchmarks

## Procedure invocation

- Return address saving conventions
  - Caller saves registers
  - Callee saves registers
- Caller save must be used if procedures can access global variables
- Most compilers enforce this

## Control flow instruction summary

- Expect PC-relative branch displacement of at least 8 bits
- Expect register indirect and PC-relative addressing for jump instructions to support returns and other features

## Checkpoint in architectural requirements at ISA level

- Load-store architecture
- Addressing modes: displacement, immediate, register indirect
- Data: 8-, 16-, 32-, 64-bit integers, 32-, 64-bit floating point
- Simple operations
- PC-relative conditional branches
- Jump
- Link instruction for procedure call
- Register indirect jumps for procedure return

## Instruction set encoding

- Opcode field
- Address specifier field
- More bits used to encode addressing modes/register fields than to specify opcode

## Competing forces in ISA encoding

- Wish to have as many registers/addressing modes as possible
- Field size influences instruction length and average programme size
- Instructions should be encoded into lengths that are easily handled in pipelines
- Instruction length a multiple of bytes rather than arbitrary bit length

## Instruction encoding variations

- Variable length
  Example: Intel 80x86, Vax
- Fixed length
  Example: Alpha, ARM, MIPS, PowerPC, SPARC, SuperH
- Hybrid
  Example: IBM 360/370, MIPS16, Thumb

## Embedded RISC

- Smaller code size important in embedded systems
- 32-bit fixed format not suitable
- Reduced-length RISC instructions: Thumb (ARM), MIPS16
- IBM compresses standard instructions and decompresses on field/cache miss
- Hitachi uses fixed 16-bit format RISC, SuperH

## Instruction set properties to aid compiler writing

- Regularity
- The three primary parts of an instruction set (operations, data types, addressing modes) should be orthogonal
- Aspects of an architecture are said to be orthogonal if they are independent
- Operations and addressing modes are orthogonal if, for every operation to which one addressing mode can be applied, all addressing modes are applicable
- Provide primitives, not solutions
- Special features that match a language construct or kernel function are often unusable
- Provide instructions that fix the quantities known at compile time as constants