Instruction Sets
Ch 9-10

Characteristics
Operands
Operations
Addressing
Instruction Formats

Instruction Set

(käskykanta)

- Collection of instructions that CPU understands
- Only interface to CPU from outside
- CPU executes a program ⇔ CPU executes given instructions "one at a time"
  - fetch-execute cycle

Fig. 9.1

Machine Instruction

Fig. 9.1

- Opcode
  - What should I do? Math? Move? Jump?
- Source operand references
  - Where is the data to work on? Reg? Memory?
- Result operand reference
  - Where should I put the result? Reg? Memory?
- Next instruction reference
  - Where is the next instruction? Default? Jump?

Instruction Representation

- Bit presentation:
  - binary program
- Assembly language
  - symbolic program
- Symbolic assembly language

0x2465A080
Opcode, operands
LOAD R1,=0x6678
Symbolic opcode
Virtual or physical address?
LOAD R1,TotalSum
Symbolic value?
Fig. 9.11

Instruction Set Design (5)

- Operation types (operaatiotyyppi)
  - How many? What type? Simple? Complex?
- Data types (tietotyyppi)
  - Just a few? Many?
- Instruction format (käskyn muoto)
  - fixed length? Varying length? Nr of operands?
- Number of addressable registers
  - too many Þ long instructions
- Addressing (tiedon osoitus)
  - What modes to use to address data and when?

Good Instruction Set (2)

- Good target to compiler
  - Easy to compile?
  - Easy to compile code that runs fast?
  - Possible to compile code that runs fast?
- Allows fast execution of programs
  - How many meaningless instructions per second?
  - How fast does my program run?
    - Solve linear system of 1000 variables?
    - Set of data base queries?

Good Instruction Set (contd) (5)

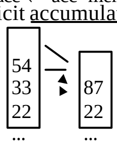
- Beautiful & Aesthetic
  - Orthogonal (ortogonaalinen)
    - Simple, no special registers, no special cases, any data type or addressing mode can be used with any instruction
  - Complete (täydellinen)
    - Lots of operations, good for all applications
  - Regular (säännöllinen)
    - Specific instruction field has always same meaning
  - Streamlined (virtaviivainen)
    - Easy to define what resources are used

Good Instruction Set (contd) (2)

- Easy to implement
  - 18 months vs. 36 months?
  - Who will be 1st in market? Who will get development monies back and who will not?
- Scalability (skaalautuva)
  - Speed up clock speed 10X, does it work?
  - Double address length, does design extend?
    - E.g., 32 bits ⇒ 64 bits ⇒ 128 bits?

Number of Operands? (4)

- 3? ADD A,B,C Mem(A) ← mem(B) + mem(C)
  - Normal case now ADD R1, R2, R3 r1 ← r2+r3
- 2? ADD R1, R2 r1 ← r1+r2
  - 1 operand and result the same
- 1? ADD A acc ← acc+mem(A)
  - 1 operand and result in implicit accumulator
- 0?
  - All operands and result in implicit stack ADD



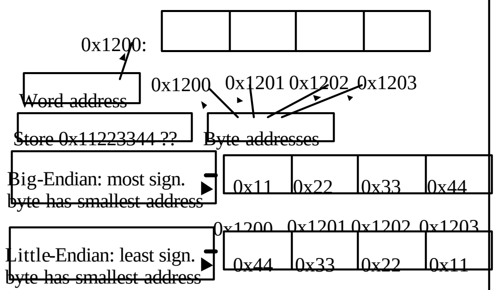
Instruction Set Architecture (ISA) Basic Classes

- Accumulator
- Stack
- General Purpose Register
  - only one type of registers, good for all
  - 2 or 3 operands
- Load/Store
  - only load/store instructions access memory
  - 3 operand ALU instructions

```
LOAD R3, C
LOAD R2,B
ADD   R1,R2,R3
STORE R1,A
```


Big vs. Little Endian (3)

- How are multi-byte values stored



Big vs. Little Endian

- Address of multi-byte data items is the same in both representations
- Only internal byte order varies
- Must decide one way or the other
  - Math circuits must know which presentation used
  - Must consider when moving data via network
- Power-PC: bi-endian - both modes at use
  - can change it per process basis
  - kernel mode selected separately

Data (Operands, Result) Location

- Register — acc, r2, r8, register stack, f4, f15
  - close, fast
  - limited number of them
  - need to load/store values from/to memory sometimes (often)
    - Big problem! 50% of compiler time to decide
    - register allocation problem
- Memory — memory stack (hw regs have mem addresses), 0x345670
  - far away
  - only possibility for large data sets
    - vectors, arrays, sets, tables, objects, ...

Aligned Data (4)

2 byte (16 bit) halfword has byte address: 0010…10010

4 byte (32 bit) word has byte address: 0010…10100

8 byte (64 bit) doubleword has byte address: 0010…11000

- Aligned data — 11 22 33 44
  - faster memory access
    - 32-bit data loaded as one memory load
- Non-aligned data — 11 22 / 33 44
  - saves mem, more bus traffic!
    - 32-bit non-aligned data requires 2 memory loads (each 4 bytes) and combining data into one 32-bit data item

Data Types (8)

- Address — 16b, 32b, 64b, 128b?
- Integer — 16b, 32b, 64b?
- Floating point — 32b, 64b, 80b?
- Decimal — 18 digits (9 bytes) packed decimal?
- Character — 1 byte = 8b, IRA = ASCII, EBCDIC?
- String — finite, arbitrary length?
- Logical data — 1 bit (Boolean value, bit field)?
- Vector, array, record, ….

Size of Operand

- 1 word, 32 bits — int, float, addr
- 2 words, 64 bits — double float, addr
- 4 words, 128 bits — addr
- 1 byte (8 bits) — char
- 2 bytes — short int
- 1 bit — logical values

Pentium II Data Types

- General data types
  - 8-bit byte
  - 16-bit word
  - 32-bit doubleword
  - 64-bit quadword
- Not aligned
- Little Endian
- Specific data types — Table 9.2
- Numerical data types — Figure 9.4

Operation Types

Table 9.3

- Data transfer
  - CPU ↔ memory
- ALU operations
  - INT, FLOAT, BOOLEAN, SHIFT, CONVERSION
- I/O
  - read from device, start I/O operation
- Transfer of control
  - jump, branch, call, return, IRET, NOP
- System control — Table 9.4
  - HALT, SYSENTER, SYSEXIT, …
  - CPUID returns current HW configuration
    - size of L1 & L2 caches, etc

Data References (2)

- Where is data?
  - in memory
  - in registers
  - in instruction itself
- How to refer to data?
  - various addressing modes
  - multi-phase data access
    - how is data location determined (addressing mode)
    - compute data address (register? effective address?)
    - access data

Addressing Modes (Ch 10)

Fig

Table. 10.1

- Immediate
- Direct — Me
- Indirect — Address of memory address of data
- Register — Data in register (best case?)
- Register Indirect — Register has memory address (pointer)
- Displacement — Addr = reg value + constant
- Stack — Data is stack pointed by some register

Displacement Address

- Effective address = (R1) + A
  - Contents of R1
  - Constant from instruction
- Constant is often small (8 bits, 16 bits?)
- Many uses
  - PC relative — JUMP -40(PC)
  - Base register address — CALL Summation(BX)
  - Array index — ADDF F2, F2, Table(R5)
  - Record field — MUL F4, F6, Salary(R8)
  - Stack references — STORE F2, -4(FP)

More Addressing Modes

size of operand

- Autoincrement — EA = (R), R ← (R) + S
  - E.g., R pointer to an array
- Autodecrement — R ← (R) - S, EA = (R)
  - E.g., R pointer to an array
- Autoincrement deferred — EA = Mem(R), R ← (R) + S
  - E.g., R pointer to an array of pointers
- Scaled — EA = A + (R_j) + (R_i) * S
  - E.g., item (R_i, R_j) in 2-dimensional array A[i,j]

Pentium II Addressing Modes

- Immediate
  - 1, 2, 4 bytes
- Register operand
  - 1, 2, 4, 8 byte registers
  - not all registers with every ins
- Operands in Memory — Fig. 10.2
  - compute effective address and combine with segment register to get linear address (virtual address)

Table 10.2

Instruction Format (4)

- How to represent instructions in memory?
- How long instruction
  - Descriptive or dense? Code size?
- Fast to load?
  - In many parts?
  - One operand description at a time?
- Fast to parse?
  - All instruction same size & same format?
  - Very few formats?

Instruction Format (contd) (3)

- How many addressing modes?
  - Fewer is better, but harder to compile to
- How many operands?
  - 3 gives you more flexibility, but takes more space
- How many registers?
  - 16 regs → need 4 bits to name it
  - 256 regs → need 8 bits to name it
  - need at least 16-32 for easy register allocation

Instruction Format (contd) (3)

- How many register sets?
  - A way to use more registers without forcing long instructions for naming them
  - One register set for each subroutine call?
  - One for indexing, one for data?
- Address range, number of bits in displacement
  - more is better, but it takes space
- Address granularity
  - byte is better, but word address is shorter

Pentium II Instruction Set (5)

- CISC - Complex Instruction Set Computer
- At most one memory address
- "Everything" is optional
- "Nothing" is fixed
- Difficult to parse
  - all latter fields and their interpretation depend on earlier fields

Fig. 10.8

Pentium II Instruction Prefix Bytes (4)

Fig. 10.8 (a)

- Instruction prefix (optional)
  - LOCK - exclusive use of shared memory
  - REP - repeat instruction for string characters
- Segment override (optional)
  - override default segment register
  - default is implicit, no need to store it every instruction
- Address size (optional)
  - use the other (16 or 32 bit) address size
- Operand size (optional)
  - use the other (16 or 32 bit) operand size

Pentium II Instruction Fields (3)

Fig. 10.8 (b)

- Opcode
  - specific bit for byte size data
- Mod r/m (optional)
  - data in reg (8) or in mem?
  - which addressing mode of 24?
  - can also specify opcode further for some opcodes
- SIB (optional)
  - extra field needed for some addressing modes
  - scale for scaled indexing
  - index register
  - base register

Pentium II Instruction Fields (contd) (2)

Fig. 10.8 (b)

- Displacement (optional)
  - for certain addressing modes
  - 1, 2, or 4 bytes
- Immediate (optional)
  - for certain addressing modes
  - 1, 2, or 4 bytes

04/10/2000 Copyright Teemu Kerola 2000 31

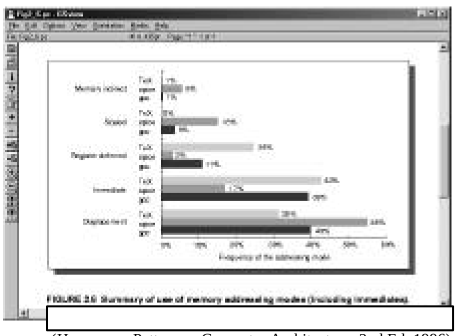
PowerPC Instruction Format (7)

Fig. 10.9

- RISC - Reduced Instruction Set Computer
- Fixed length, just a few formats
- Only load/store instructions access memory
- Only 2 addressing modes for data
- 32 general purpose registers can be used everywhere
- Fixed data size
  - no string ops
- Simple branches
  - CR-field determines which register to compare
  - L-bit determines whether a subroutine call
  - A-bit determines if branch is absolute or PC-relative

04/10/2000 Copyright Teemu Kerola 2000 32

-- End of Chapters 9-10: Instruction Sets --



(Hennnessy -Patterson, Computer Architecture, 2nd Ed, 1996)

04/10/2000 Copyright Teemu Kerola 2000 33