

Instruction Sets

Ch 10-11

Characteristics

Operands

Operations

Addressing

Instruction Formats

24.9.2002

Copyright Teemu Kerola 2002

1

Instruction Set

(käskykanta)

- Collection of instructions that CPU understands
- Only interface to CPU from outside
- CPU executes a program \Leftrightarrow CPU executes given instructions “one at a time”
 - fetch-execute cycle

Fig. 10.1 (Fig. 9.1 [Stal99])

24.9.2002

Copyright Teemu Kerola 2002

2

Machine Instruction

Fig. 10.1 (Fig. 9.1 [Stal99])

- 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?

24.9.2002

Copyright Teemu Kerola 2002

3

Instruction Representation

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

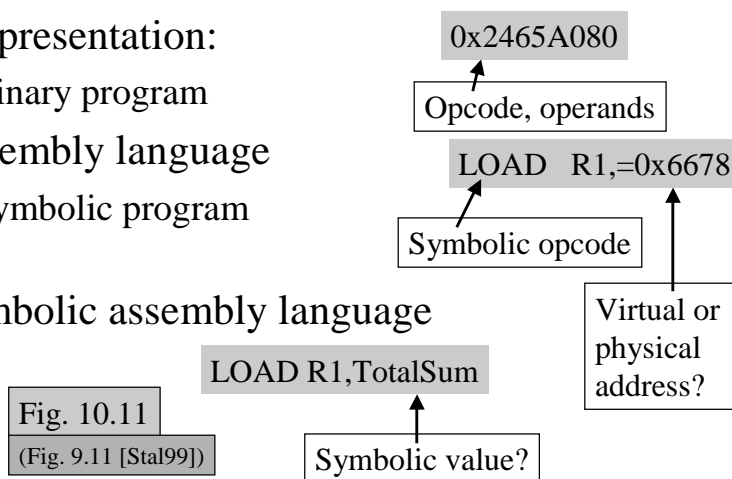


Fig. 10.11 (Fig. 9.11 [Stal99])

24.9.2002

Copyright Teemu Kerola 2002

4

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 \Rightarrow too long instructions?
 - too few \Rightarrow too hard to optimise code?
- Addressing (tiedon osoitus)
 - What modes to use to address data and when?

24.9.2002

Copyright Teemu Kerola 2002

5

Good Instruction Set ⁽²⁾

- Good target for compiler
 - Easy to compile?
 - Possible to compile code that runs fast?
 - Easy to compile code that runs fast?
- Allows fast execution of programs
 - How many meaningless instructions per second? MIPS? GFLOPS?
 - How fast does my program run?
 - Solve linear system of 1000 variables?
 - Set of data base queries?
 - Connect a phone call in reasonable time?

24.9.2002

Copyright Teemu Kerola 2002

6

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

24.9.2002

Copyright Teemu Kerola 2002

7

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 \Rightarrow 64 bits \Rightarrow 128 bits?

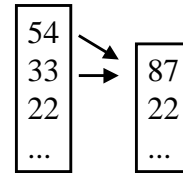
24.9.2002

Copyright Teemu Kerola 2002

8

Number of Operands? ⁽⁴⁾

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



24.9.2002

Copyright Teemu Kerola 2002

9

Instruction Set Architecture (ISA) Basic Classes

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

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

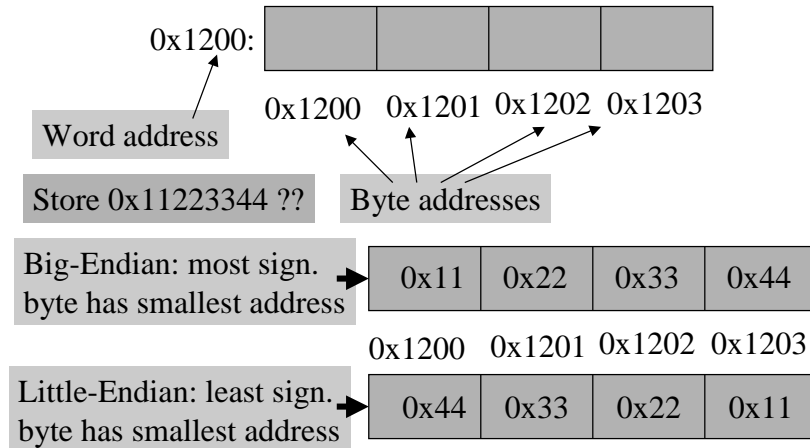
24.9.2002

Copyright Teemu Kerola 2002

10

Big vs. Little Endian ⁽³⁾

- How are multi-byte values stored



24.9.2002

Copyright Teemu Kerola 2002

11

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
 - Little-Endian may be faster
 - 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

24.9.2002

Copyright Teemu Kerola 2002

12

Data (Operands, Result) Location

- Register
 - 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
 - far away
 - only possibility for large data sets
 - vectors, arrays, sets, tables, objects, ...

acc r2, r8
 register stack f4, f15

memory stack
 (hw regs have mem addresses)

memory 0x345670
 cache?

24.9.2002

Copyright Teemu Kerola 2002

13

Aligned Data ⁽⁴⁾

2 byte (16-bit) half-word 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
 - faster memory access
 - 32-bit data loaded as one memory load
- Non-aligned data
 - 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

11	22	33	44
----	----	----	----

		11	22
--	--	----	----

33	44		
----	----	--	--

24.9.2002

Copyright Teemu Kerola 2002

14

Data Types ⁽⁸⁾

- Address 16b, 32b, 64b, 128b?
- Integer 16b, 32b, 64b?
- Floating point 32b, 64b, 82b?
- Decimal 18 digits (9 bytes) packed decimal?
- Character 1 byte = 8b IRA = ASCII, EBCDIC?
- String finite, arbitrary length? Length denotation?
- Logical data 1 bit (Boolean value, bit field)?
- Vector, array, record,

24.9.2002

Copyright Teemu Kerola 2002

15

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

24.9.2002

Copyright Teemu Kerola 2002

16

Example: 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 10.2 (Tbl. 9.2 [Stal99])
(for Pentium II)
- Numerical data types Figure 10.4 (Fig. 9.4 [Stal99])

24.9.2002

Copyright Teemu Kerola 2002

17

Operation Types

- Data transfer Table 10.3
(Tbl 9.3 [Stal99])
 - 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
 - HALT, SYSENTER, SYSEXIT, ...
 - CPUID returns current HW configuration Table 10.4
(Tbl 9.4 [Stal99])
 - size of L1 & L2 caches, etc

24.9.2002

Copyright Teemu Kerola 2002

18

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

24.9.2002

Copyright Teemu Kerola 2002

19

24.9.2002

Copyright Teemu Kerola 2002

20

Addressing Modes (Ch 10)

Fig. 11.1

(Fig 10.1 [Stal99])

Table. 11.1

(Tbl 10.1 [Stal99])

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

24.9.2002

Copyright Teemu Kerola 2002

21

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)

24.9.2002

Copyright Teemu Kerola 2002

22

More Addressing Modes

- Autoincrement $EA = (R), R \leftarrow (R) + S$
E.g., `CurrIndex = i++;`
- Autodecrement $R \leftarrow (R) - S, EA = (R)$
E.g., `CurrIndex = --i;`
- Autoincrement deferred $EA = Mem(R), R \leftarrow (R) + S$
E.g., `Sum = Sum + (*ptrX++);`
- Scaled $EA = A + (R_j) + (R_i) * S$
E.g., `double X;`
`X = Tbl[i][j];`

size of operand
↓

24.9.2002

Copyright Teemu Kerola 2002

23

Pentium II Addressing Modes

- Immediate
 - 1, 2, 4 bytes
- Register operand
 - 1, 2, 4, 8 byte registers
 - not all registers with every instruction
- Operands in Memory Fig. 11.2 (Fig 10.2 [Stal99])
 - compute effective address and combine with segment register to get linear address (virtual address)

Table 11.2 (Tbl 10.2 [Stal99])

24.9.2002

Copyright Teemu Kerola 2002

24

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 (I.e., split into logical components)?
 - All instruction same size & same format?
 - Very few formats?

24.9.2002

Copyright Teemu Kerola 2002

25

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
 - How many registers, that can be referenced in one instruction vs. referenced overall?

24.9.2002

Copyright Teemu Kerola 2002

26

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

24.9.2002

Copyright Teemu Kerola 2002

27

Pentium 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. 11.8 (Fig 10.8 [Stal99])

24.9.2002

Copyright Teemu Kerola 2002

28

Pentium Instruction

Prefix Bytes ⁽⁴⁾

Fig. 11.8

- Instruction prefix (optional) (Fig 10.8 (a)[Stal99])
 - 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

24.9.2002

Copyright Teemu Kerola 2002

29

Pentium Instruction Fields ⁽³⁾

Fig. 11.8

- Opcode (Fig 10.8 (a)[Stal99])
 - 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) – Scale/Index/Base
 - extra field needed for some addressing modes
 - scale for scaled indexing
 - index register
 - base register

24.9.2002

Copyright Teemu Kerola 2002

30

Pentium Instruction Fields

(contd) (2)

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

Fig. 11.8

(Fig 10.8 (a)[Stal99])

24.9.2002

Copyright Teemu Kerola 2002

31

PowerPC Instruction Format (7)

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

Fig. 11.9

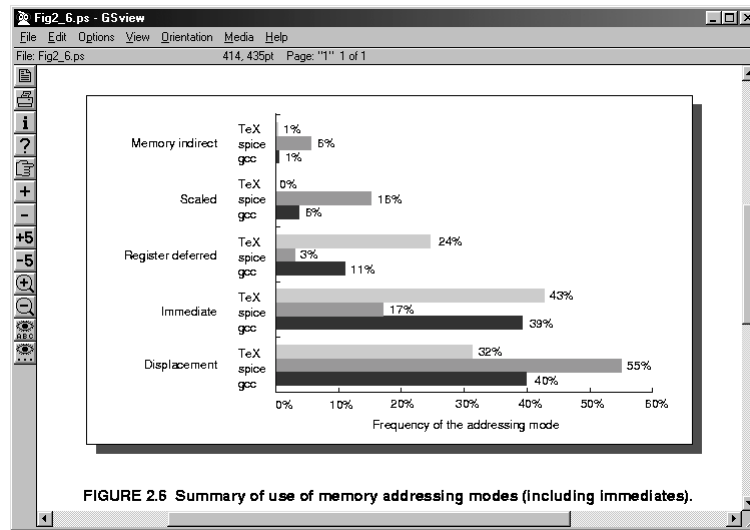
(Fig 10.9 [Stal99])

24.9.2002

Copyright Teemu Kerola 2002

32

-- End of Chapters 10-11: Instruction Sets --



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