Instruction sets (Käskykannat)

Ch 10-11 [Sta10]
- Operations
- Operands
- Operand references (osoitustavat)
- Pentium / ARM

Computer Instructions (konekäskyt)

- Instruction set (käskykanta) = Set of instructions CPU 'knows'
- Operation code (käskykoodi)
  - What does the instruction do?
- Data references (viitteet) – one, two, several?
  - Where does the data come for the instruction?
    - Registers, memory, disk, I/O
  - Where is the result stored?
    - Registers, memory, disk, I/O
  - What instruction is executed next?
    - Implicit? Explicit?
  - I/O?
    - Memory-mapped I/O → I/O with memory reference operations

Instructions and data (käskyt ja data)

<table>
<thead>
<tr>
<th>Address</th>
<th>Content</th>
<th>Address</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>000</td>
<td>000</td>
<td>000</td>
</tr>
<tr>
<td>001</td>
<td>000</td>
<td>001</td>
<td>000</td>
</tr>
<tr>
<td>010</td>
<td>000</td>
<td>010</td>
<td>000</td>
</tr>
<tr>
<td>011</td>
<td>000</td>
<td>011</td>
<td>001</td>
</tr>
<tr>
<td>100</td>
<td>000</td>
<td>100</td>
<td>001</td>
</tr>
<tr>
<td>101</td>
<td>000</td>
<td>101</td>
<td>001</td>
</tr>
<tr>
<td>110</td>
<td>000</td>
<td>110</td>
<td>001</td>
</tr>
<tr>
<td>111</td>
<td>000</td>
<td>111</td>
<td>001</td>
</tr>
</tbody>
</table>

Instruction types?
- Transfer between memory and registers
  - LOAD, STORE, MOVE, PUSH, POP, ...
- Controlling I/O
  - Memory-mapped I/O (like memory)
  - I/O not memory-mapped – own instructions to control
- Arithmetic and logical operations
  - ADD, MUL, CLR, SET, COMP, AND, SHR, NOP, ...
- Conversions (esim. tapaamistapa)
  - TRANS, CONV, 16To32, IntToFloat, ...
- Transfer of control (käskyjen suoritusjärjestyksen ohjaus), conditional, unconditional
  - JUMP, BRANCH, CALL, EXIT, HALT, ...
- Service requests (palvelupyyntö)
  - SVC, INT, SYSENTER, SYSEXIT, ...
- Privileged instructions (etuoikeutetut käskyt)
  - DIS, IEN, flush cache, invalidate TUB, ...

What happens during instruction execution?

- Transfer data from one location to another:
  - If memory is involved:
    - Determine memory address
    - Perform virtual-to-actual memory address transformation
    - Check cache
    - Initialize memory read/write
  - May involve data transfer; before and/or after
  - Perform function in ALU
  - Set condition codes and flags
- Logical
  -Same as arithmetic
- Conversion:
  - Similar to arithmetic and logical. May involve special logic to perform conversion
- Transfer of Control:
  - Update program counter. For subroutine collection, manage parameter passing and locals
  - Same command as I/O module
- I/O
  - If memory-mapped I/O, determine memory-mapped address
Lecture 6: Instruction sets  
10.11.2010

What kind of data?
- Integers, floating points
- Boolean (tottuusarvoja)
- Characters, strings
- IPA (aka ASCII), EBCDIC
- Vectors, tables
- N elements in sequence
- Memory references
- Different sizes
  - 8 / 16 / 32 / 64b, ...
  - Each type and size has its own operation code

Instruction representation (käskyformaatti)
- How many bits for each field in the instruction?
- How many different instructions?
- Maximum number of operands per instruction?
- Operands in registers or in memory?
- How many registers?
- Fixed or variable size (vakio vai vaihteleva koko)?

Architectures
- Accumulator-based architecture (akkukone)
  - Just one register, accumulator, implicit reference to it
- Stack-based (pinokone)
  - Operands in stack, implicit reference
  - PUSH, POP
- Register-based (yleisrekisterikone)
  - All registers of the same size
  - Instructions have 2 or 3 operands
- Load/Store architecture
  - Only LOAD/STORE have memory refs
  - ALU-operations have 3 regs

Byte ordering (tavujärjestys): Big vs. Little Endian
- How to store a multibyte scalar value?

<table>
<thead>
<tr>
<th>Operation</th>
<th>Microcode</th>
<th>Name</th>
<th>Number of Bits Transferred</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>Load</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>LR</td>
<td>Load High (word)</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>LR</td>
<td>Load</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>LER</td>
<td>Load (word)</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>LE</td>
<td>Load</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>LDR</td>
<td>Load (Long)</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>LD</td>
<td>Load</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>ST</td>
<td>Store</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>STRH</td>
<td>Store High (word)</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>STRC</td>
<td>Store Counter</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>STS</td>
<td>Store Start (word)</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>STD</td>
<td>Store (Long)</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

IBM ESA/390

How many registers?
- Minimum 16 to 32
- Work data in registers
- Different register (sets) for different purpose?
  - Integers vs. floating points, indices vs. data, code vs. data
  - All sets can start register numbering from 0
  - Opcode determines which set is used
- More registers than can be referenced?
  - CPU allocates them internally
    - Register window = virtual register names
    - Example: function parameters passed in registers
    - Programmer thinks that registers are always r8-r15,
    - CPU maps r8-r15 somewhere to r8-r132
    - (We’ll come back to this later)

Byte ordering (tavujärjestys)
- Big vs. Little Endian

Big: 0x00000044
Little: 0x44000000

See: Appendix 10B (Sta10)

Big vs. Little Endian

ARM uses only one of them
- Little-endian: x86, Pentium, VAX
- Big-endian: IBM 370, Motorola 68000 (Mac),
  most RISC-architectures
- ARM, a bi-endian machine, accepts both
  - System control register has 1 bit (E-bit) to indicate the endian mode
  - Program controls which to use
- Byte order must be known, when transferring data from one machine to another
  - Internet uses big-endian format
  - Socket library (pistokekirjasto) has routines htons and htonl
  - (Host to Internet & Internet to Host)
**Data alignment (kohdentaminen)**

- 16b data starts with even (parillinen) (byte)address
- 32b data starts with address divisible (jaollinen) by 4
- 64b data starts with address divisible by 8
- Aligned data is easier to access
- 32b data can be loaded by one operation accessing the word address (sanaosoite)
- Unaligned data would contain no ‘wasted’ bytes, but
  - For example, loading 32b unaligned data requires two loads from memory (word address) and combining it

**Aligned data is easier to access**

**32b data can be loaded by one operation accessing the word address (sanaosoite)**

Unaligned data would contain no ‘wasted’ bytes, but
- For example, loading 32b unaligned data requires two loads from memory (word address) and combining it

**Memory references (Muistin osoitustavat)**

Ch 11 [Sta10]

Where are the operands?

- In the memory
  - Variable of the program, stack (pino), heap (keko)
- In the registers
  - During the instruction execution, for speed
- Directly in the instruction
  - Small constant values
- How does CPU know the specific location?
  - Bits in the operation code
  - Several alternative addressing modes allowed

**Addressing modes (osoitusmuodot)**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Algorithm</th>
<th>Principal Advantage</th>
<th>Principal Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instruction</td>
<td>Operated = A</td>
<td>No memory reference</td>
<td>Limited operand magnitude</td>
</tr>
<tr>
<td>Direct</td>
<td>EA = A</td>
<td>Simple</td>
<td>Large address space</td>
</tr>
<tr>
<td>Indirect</td>
<td>EA = (A)</td>
<td>Large memory references</td>
<td>Multiple memory references</td>
</tr>
<tr>
<td>Register</td>
<td>EA = R</td>
<td>No memory reference</td>
<td>Limited address space</td>
</tr>
<tr>
<td>Register indirect</td>
<td>EA = (R)</td>
<td>Large address space</td>
<td>Entry memory reference</td>
</tr>
<tr>
<td>Displacement</td>
<td>EA = A + b</td>
<td>Flexibility</td>
<td>Complexity</td>
</tr>
<tr>
<td>Stack</td>
<td>EA = top of stack</td>
<td>No memory reference</td>
<td>Limited applicability</td>
</tr>
</tbody>
</table>

**Addressing modes**

- EA = Effective Address
- (A) = content of memory location A
- (R) = content of register R
- One register for the top-most stack item’s address
- Register (or two) for the top stack item (or two)

**Displacement Address (siirrymä)**

- Effective address = (R1) + A
  - register content + constant in the instruction
- Constant relatively small (8 b, 16 b?)
- Usage
  - JUMP 9+5
  - CALL SP, Summation(BX)
  - ADDF F2, F2, Table(R5)
  - MUL F4, F6, Salary(R8)
  - STORE F2, -4(FP)
More addressing modes

- **Autoincrement (before/after)**
  - Example: `CurriXdecr = ++X`
  - `EA = (R), R = (R) + S`

- **Autodecrement (before/after)**
  - Example: `CurriXdecr = --X`
  - `EA = (R), R = (R) - S`

- **Autoincrement deferred**
  - Example: `Sum = Sum + (*ptrX++)`
  - `EA = Mem[R], R = (R) + S`

- **Autoscale**
  - Example: `Double X; ...
    X = Tbl[i];`
  - `EA = A + (R) * S`

---

**Pentium:**

**Registers:**

- General registers (`yleisrekisterit`):
  - EAX, EBX, ECX, EDX (accumulator, base, count, data)
  - ESI, EDI (source & destination index)
  - ESP, EBP (stack pointer, base pointer)
- Segment registers 16b:
  - CS, SS, DS, ES, FS, GS - code, stack, data, ...
  - ESP, EBP (stack pointer, base pointer)
- **Program counter (käskynosoitin):**
  - **EP Extended Instruction Pointer**
- Status register:
  - **EFLAGS**:
    - overflow, sign, zero, parity, carry, ...

---

**Pentium: Operations**

- Data transfers, arithmetic, moves, jumps, stores, etc.
- **MMX Operations**
  - No under/overflow.
  - Use closest representation
  - **SIMD**

---

**Computer Organization II**

**Data types**

- General: Byte, word (32-bit), doubleword (64-bit), quadword (64-bit), and quadword (128-bit for MMX-related) memory counters.
- Pointer: A 32-bit linear value contained in a type word, or holder in a type two-component representation.
- Signed: An unsigned quantity contained in a byte, word, or doubleword.
- Unaligned: The aligned value is the first word of the alignment. The byte size of such a word is 1, 2, 4, or 8 bits.
- Unaligned: The aligned value is the first word of the alignment. The byte size of such a word is 1, 2, 4, or 8 bits.
- Signed: A 32-bit signed integer, containing the maximum number of bits.
- ...
Pentium: Addressing modes
(muistin osoitustavat)

<table>
<thead>
<tr>
<th>x86 Addressing Mode</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Immediate</td>
<td>operand = A</td>
</tr>
<tr>
<td>Register Offset</td>
<td>operand = (R)</td>
</tr>
<tr>
<td>Displacement</td>
<td>LA = (SR) + A</td>
</tr>
<tr>
<td>Base</td>
<td>LA = (SR)</td>
</tr>
<tr>
<td>Base with Displacement</td>
<td>LA = (SR) + (B) + A</td>
</tr>
<tr>
<td>Scale Index with Displacement</td>
<td>LA = (SR) + (I) x S + (B) + A</td>
</tr>
<tr>
<td>Relative Base</td>
<td>LA = (PC) + A</td>
</tr>
</tbody>
</table>

- LA = linear address
- NC = constants of X
- SR = segment register
- PC = program counter
- B = base register
- I = index register
- S = scaling factor
- A = contents of an address field in the instruction

(Pentium Table 11.2)

Pentium: Instruction format

- CISC
  - Complex Instruction Set Computer
- Lots of alternative fields only op-code always?
  - Part may be present or absent in the bit sequence
  - Prefix 0-4 bytes
  - Interpretation of the rest of the bit sequence depends on the content of the preceding fields
- Plenty of alternative addressing modes (osoitustavat)
  - At most one operand can be in the memory
  - 24 different
- Backward compatibility
  - OLD 16-bit 8086-programs must still work
    - How to handle old instructions: emulate, simulate?

Pentium: Instruction format

- Instruction prefix (optional)
  - LOCK – exclusive use of shared memory in multiprocessor env.
  - REP – repeat operation to all characters of a string
- Segment override (optional)
  - Use the segment register explicitly specified in the instruction
  - Else use the default segment register (implicit assumption)
- Operand size override (optional)
  - Switch between 16 or 32 bit operand, override default size
- Address size override (optional)
  - Switch between 16 or 32 bit addressing. Override the default, which could be either

Pentium: Instruction format

- Opcode
  - Each instruction has its own bit sequence (incl. opcode)
  - Bits specify the size of the operand (8/16/32b)
- ModR/m (optional)
  - Indicate, whether operand is in a register or in memory
  - What addressing mode (osoitustapa) to be used
  - Sometimes enhance the opcode information (with 3 bits)
- SIB = Scale/Index/Base (optional)
  - Some addressing modes need extra information
  - Scale: scale factor for indexing (element size)
  - Index: index register (number)
  - Base: base register (number)
Pentium: Instruction format

- Displacement (optional)
  - Certain addressing modes need this
  - 0, 1, 2 or 4 bytes (0, 8, 16 or 32 bits)
- Immediate (optional)
  - Certain addressing modes need this, value for operand
  - 0, 1, 2 or 4 bytes

ARM: Instruction set (käskykanta)

- RISC
  - Reduced Instruction Set Computer
- Fixed instruction length (32b), regular format
  - All instructions have the condition code (4 bits)
- Small number of different instructions
  - Instruction type (3 bits) and additional opcode /modifier (5 bit)
  - Easier hardware implementation, faster execution
- Thump instruction set uses 16 bit instructions

ARM Data Types

- 8 (byte), 16 (halfword), 32 (word) bits - word aligned
- Unsigned integer and twos-complement signed integer
- Majority of implementations do not provide floating-point hardware
- Little and Big Endian supported
  - Bit E in status register defines which is used

ARM Addressing mode

- Data Processing instructions
  - Register addressing
    - Value in register operands may be scaled using a shift operator
  - Or mixture of register and immediate addressing
- Branch instructions
  - Immediate
  - Instruction contains 24 bit addressing
  - Shifted 2 bits left
    - On word boundary
    - Effective range +/-32MB from PC.
**ARM Load/Store Multiple Addressing**

- Load/store subset of general-purpose registers
- 16-bit instruction field specifies list of registers
- Sequential range of memory addresses
- Base register specifies main memory address

**ARM Instruction Formats**

- For data processing instructions, updates condition codes
- For load/store multiple instructions, execution restricted to supervisor mode
- \( P, U, W \) distinguish between different types of addressing mode
- \( B \) = Unsigned byte (\( B=1 \)) or word (\( B=0 \)) access
- \( L \) = For load/store instructions, Load (\( L=1 \)) or Store (\( L=0 \))
- \( L \) = For branch instructions, is return address stored in link register

**ARM Condition codes**

- **N** – Negative
- **Z** – Zero
- **C** – Carry
- **V** – Overflow

**RISC vs. CISC**

- **RISC**
  - Easy to execute
  - Support high-level languages
difficult to execute

- **CISC**
  - Support high-level language
  - Difficult to execute

**Summary**

- Instruction set types: Stack, register, load-store
- Data types: Int, fixat, char
- Addressing modes: Indexed, others?
- Operation types?
  - Arithmetic & logical, shifts, conversions, vector
  - Comparisons
  - Control
    - If-then-else, loops, function calls/returns
    - Conditional instructions
  - Loads/stores, stack ops, vector ops
  - Privileged, os instructions
- Instruction formats
- Intel and Arm case studies

**Review Questions / Kertauskysymyksiä**

- Fields of the instruction?
- How does CPU know if the integer is 16 b or 32 b?
- Meaning of Big-Endian?
- Benefits of fixed instruction size vs. variable size instruction format?