DEPARTMENT OF
NETWORKED SYSTEMS
AND SERVICES

COMPUTER ARCHITECTURES

Instruction set architectures

Gábor Horváth, ghorvath@hit.bme.hu

Budapest,
04/27/2023

MŰEGYETEM 1782

© Department of Networked Systems and Services

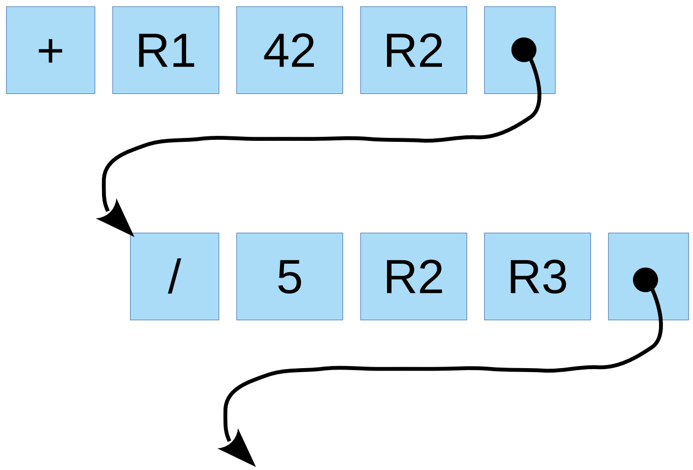
INSTRUCTION SET ARCHITECTURES

- Every processor has
  - Instruction Set Architecture (ISA)
  - A kind of programming interface
- Parts of ISA:
  - List of supported instructions
  - List of supported data types
  - Registers
  - Addressing modes
  - Flags
  - How to communicate with I/O devices
  - Interrupt and exception handling
  - Etc.

© Department of Networked Systems and Services

PROPERTIES OF THE INSTRUCTIONS

- „Parts" of an instruction:
  - Code of the operation
  - Values/addresses of operands
  - Address where the result is stored
  - Pointer to the next instruction



- For simplicity:
  - Pointer to the next instruction is unnecessary
    - The next instruction is always the next one in the memory
  - Number of operands
    - Support for 3 operands: `R1 ← R2 + R3`
    - Support for 2 operands: `R1 ← R1 + R2`
    - Support for 1 operands: `ADD R1`

PROPERTIES OF THE INSTRUCTIONS

- Instruction types:
  - Data movement
    `R1←R2, R1←MEM[100], R1←42, MEM[100]←R1, MEM[100]←42`
  - Arithmetic-logic operations
    `R1 ← R2+R3, R1 ← MEM[100]*42, MEM[100] ← R1 & R2`
  - Control flow operations:
    `JUMP -42, JUMP +28 IF R1==R2, CALL proc, RETURN`
  - Stack operations
    `PUSH R1, PUSH 42, R2 ← POP`
  - I/O operations
    `IO[42] ← R1, R1 ← IO[42]`
  - Transcendent operations
    `R2 ← SIN R1, R2 ← SQRT 42`
  - Etc.

© Department of Networked Systems and Services

ADDRESSING MODES

- Determines, where the operand is located
- Possible locations:
  - embedded in the instruction (immediate)
  - in a register
  - in the memory

Addressing mode	Example
Register	`R1 ← R2 + R3`
Immediate	`R1 ← R2 + 42`
Direct	`R1 ← R2 + MEM[42]`
Register indirect	`R1 ← R2 + MEM[R3]`
Indirect with offset	`R1 ← R2 + MEM[R3+42]`
Memory indirect	`R1 ← R2 + MEM[MEM[R3]]`
Indexed	`R1 ← R2 + MEM[R3+R4]`

CONTROL-FLOW OPERATIONS

- Typically relative addressing, e.g. JUMP -28
- 3 possible implementations of conditional branches:
  - With condition codes:
    ```
    COMPARE R1, R2
    JUMP label IF GREATER
    ```
  - With condition registers:
    ```
    R0 ← R1 > R2
    JUMP label IF R0
    ```
  - „Compare and jump”:
    ```
    JUMP label IF R1 > R2
    ```

BRANCH PREDICATION

- Predicates: instructions with conditions

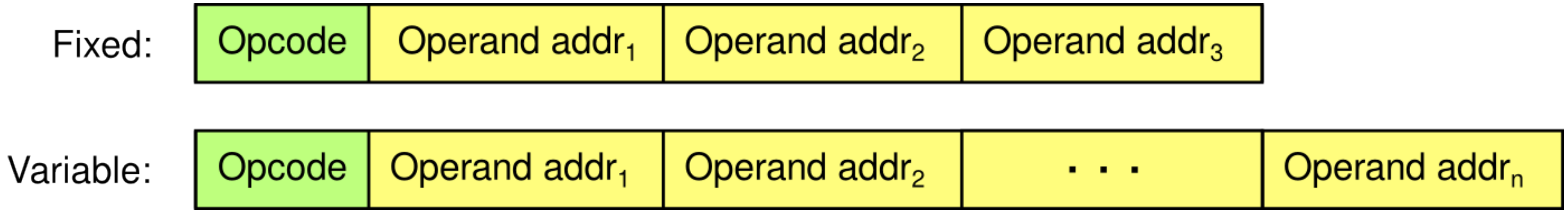
  R1 ← R2 + 32 IF P2

- Predicate registers: 1-bit registers (**P2** in the example)
- Setting the predicate registers: by comparison instructions

  P2 ← R3 ≤ R5

PROPERTIES OF THE INSTRUCTIONS

- The instructions are stored with a binary encoding
- Based on the length of the encoded instructions we have:
  - Fixed length encoding
  - Variable length encoding



ASSEMBLY PROGRAMMING

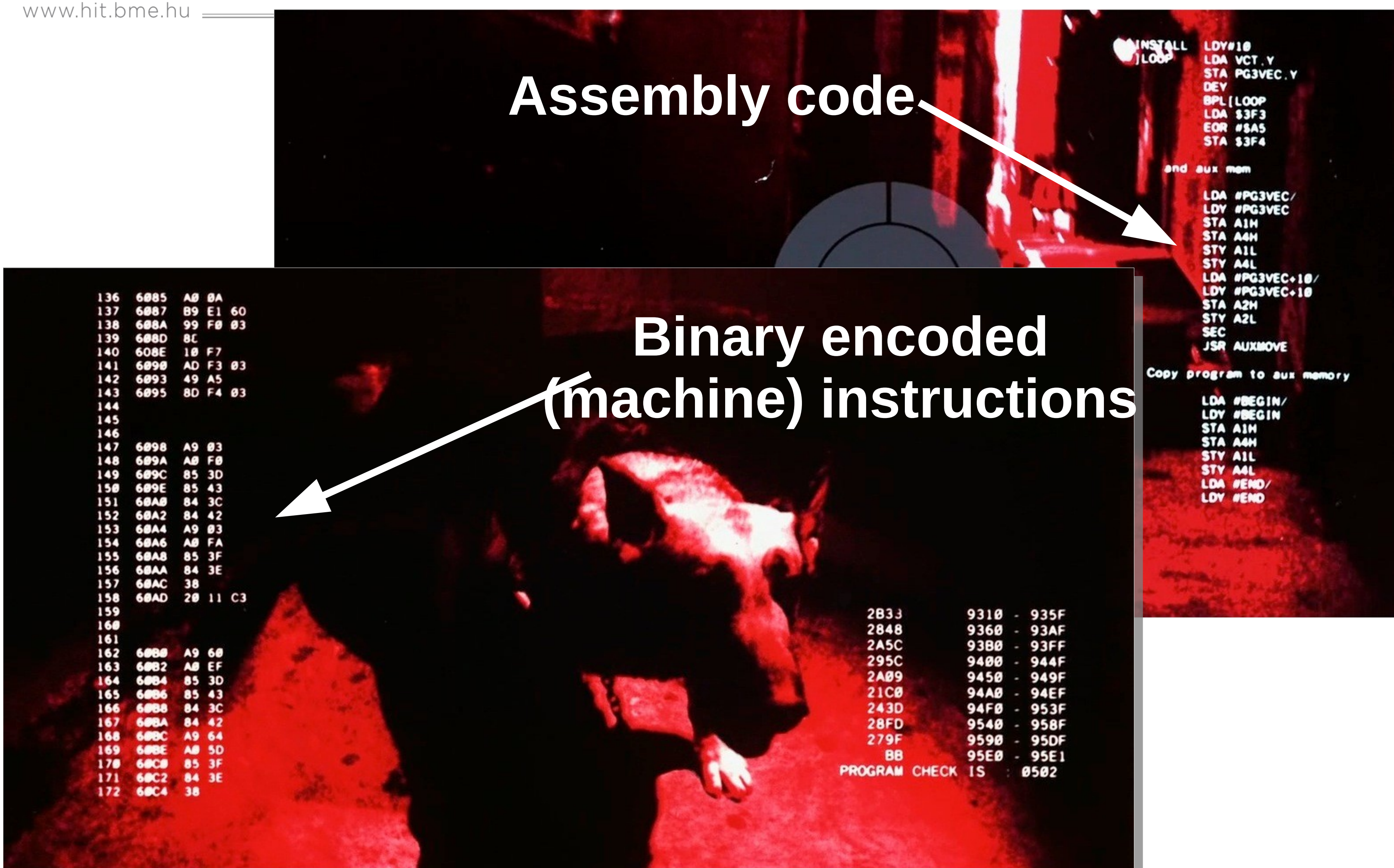
- Low-level programming:
  - Manual encoding of the instructions is inconvenient
  - Binary coded instructions are not for human use
  - Tool: assembly programming
- Assembly
  - The lowest level programming language
  - The text representation of the machine level instructions
    - 1 assembly „instruction” → 1 machine level instruction
- Assembler: creates machine code from assembly
- It is different for every instruction set architecture

ENCODING THE INSTRUCTIONS

ENCODING THE INSTRUCTIONS

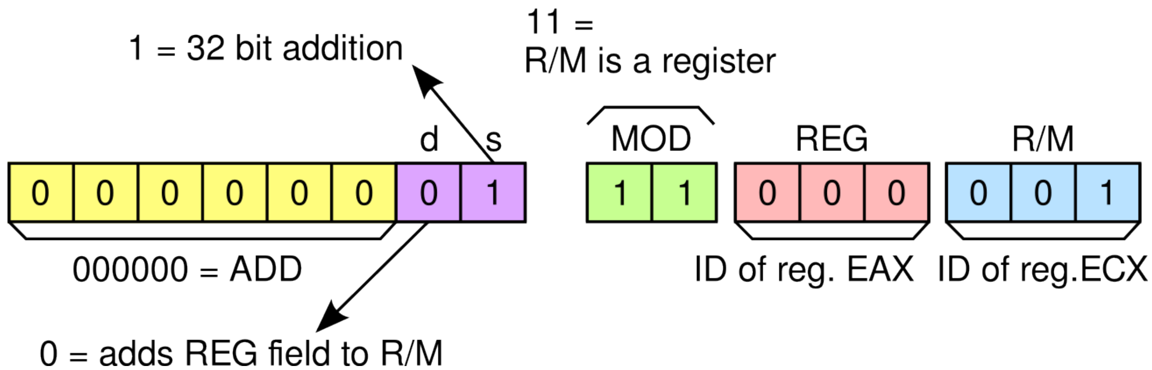
- The CPU of the Terminator: MOS-6502 (like the CPU of Apple II)
  (the cheapest CPU between 1975 and 1980: it costed sixth the price of the Intel and Motorola CPU-s at the same level of performance)
- The Terminator runs an example code of the Nibble magazine

ENCODING THE INSTRUCTIONS



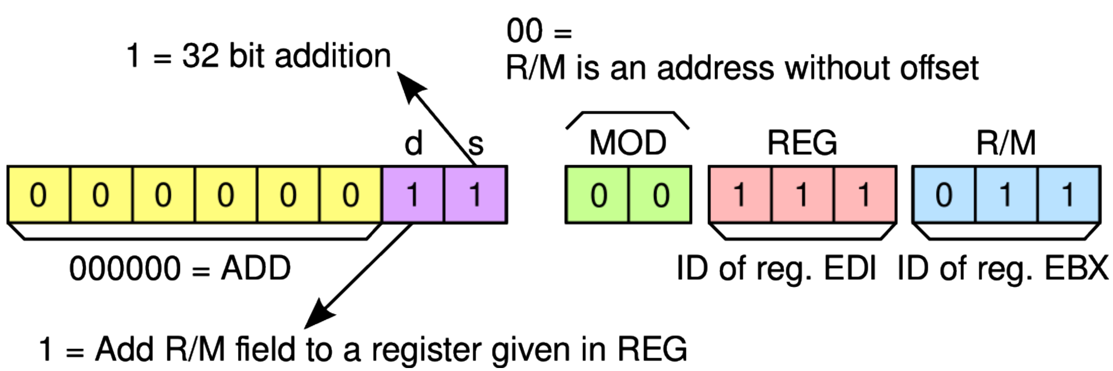
EXAMPLE FOR INSTRUCTION CODING (X86)

- ADD ECX, EAX (we write it: ECX ← ECX + EAX)



= 01 C1 (ASCII: ☺┴)

- ADD EDI, [EBX] (we write it: EDI ← EDI + MEM[EBX])

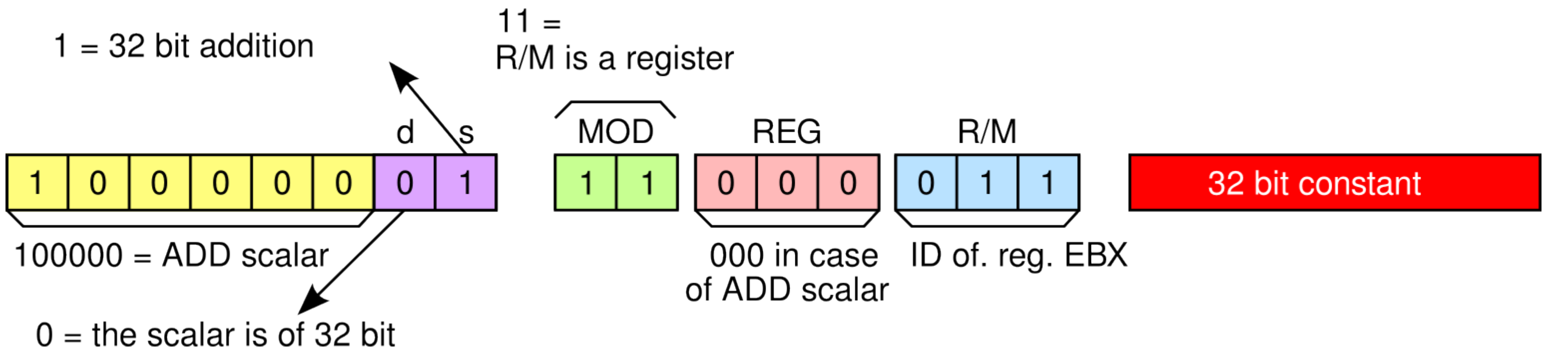


= 03 3B (ASCII: ❤ ;)

© Department of Networked Systems and Services

EXAMPLE FOR INSTRUCTION CODING (X86)

- ADD EBX, 23423765 (we write it: EBX ← EBX + 23423765)



- = 81 C3 15 6B 65 01 (ASCII: Qü§ke☺)

FURTHER PROPERTIES OF INSTRUCTION SETS

- Byte order:
  - Little endian: numbers start with the **least** significant byte
  - Big endian: numbers start with the **most** significant byte
  - Bi endian: can be selected (HW or SW)
  - Example: 23423765 (=1656B15)
    - Little endian: 15 6B 65 01
    - Big endian: 01 65 6B 15
- Ways of communicating with I/O devices:
  - Separate instructions for I/O
  - Memory mapped

© Department of Networked Systems and Services

RISC VS. CISC

- Typical in the 70's:
  - a large number of instructions
  - complex instructions
- Motivations:
  - Memory was slow – with complex instructions, the processor got more work with a single memory operation
  - Memory was expensive – a single instruction describes more work for the processor
  - Compilers were very basic that time. The processor had a „high level” like instruction set to allow easy assembly programming.
- This is the **CISC** (Complex Instruction Set Computer) philosophy
- Features:
  - Easy to use instructions
  - Register-memory instructions (pl. R1 $\leftarrow$ R2 + MEM[42])
  - Redundancy
  - Several addressing modes
  - Variable length instruction encoding
  - The execution time of the various instructions is different

RISC VS. CISC

- Typical CPU design trends in the 80's and 90's:
  - The number of instructions is as small as possible
  - The instructions are very basic
- Motivations:
  - To simplify the design of the CPU-s
  - The simpler CPU design allows more efficient implementation
- This is the **RISC** (Reduced Instruction Set Computing) philosophy
- Features:
  - Simple, elementary instructions, avoiding redundance
  - Load-Store and register-register operations
  - instead of `R1←R2+MEM[42]` we have `R3←MEM[42]; R1←R2+R3`.
  - Only a few addressing modes are available
  - Fixed instruction encoding
  - Execution time of the instructions usually takes only 1 cycle

RISC VS. CISC

- Comparison:
  - CISC: dense
    - The program is smaller
  - RISC: simple
    - Less design bugs
    - The IC is smaller
    - It consumes less energy
    - Better yield when manufacturing it
    - There is a lot of space left on the IC allowing the integration of further devices onto the same silicon
  - CISC: a small number of registers vs. RISC: much more registers

© Department of Networked Systems and Services

Some important instruction set architectures

SOME IMPORTANT INSTRUCTION SET ARCHITECTURES

x86

- First appearance: 1971, Intel 8086
- 1981: The Intel 8088 is selected as the CPU of the IBM PC
- Originally it was a 16 bit ISA, but has been extended to 32 and 64 bit later
- Nowadays it is used both in high-performance servers and low-power mobile devices
- A very obsolete ISA, but the demand for software compatibility keeps it alive for >50 years
- Intel has spent most of its profit to develop more efficient semiconductor production technology

© Department of Networked Systems and Services

SOME IMPORTANT INSTRUCTION SET ARCHITECTURES

ARM

- First implementation: 1987
- The most wide-spread ISA all over the world
- It is 32 bit right from the beginning (extension to 64 bit is in the works)
- This ISA is very carefully designed, easy to implement
  - Can be implemented with only 30.000 transistors!
- ARM does not manufacture CPUs
  - The ISA can be licensed
  - ARM designs CPUs, that can be licensed as well (ARM Cortex family)
- Primary goals: simplicity, energy efficiency
  (**not** the raw computational power)

SOME IMPORTANT INSTRUCTION SET ARCHITECTURES

POWER

- Defined in 1991 by IBM, Apple and Motorola
- Goal: to surpass the computational performance of x86
- They succeeded:
  - Huge memory and I/O bandwidth
  - 2014: 5 GHz, 12 cores, 8 threads/core (POWER8)
- Did not get popular in PCs
- But it got popular in workstations and servers
- … and all prev. gen. game consols used POWER processors! (Microsoft XBox 360, Sony PlayStation 3, Nintendo Wii)

SOME IMPORTANT INSTRUCTION SET ARCHITECTURES

SPARC

- 1987, SUN
- 64 bit from the beginning
- Open platform!
- The design of UltraSPARC T1 and T2 can be accessed by anybody (at VeriLog level)
- Still in production (now by Oracle)
  - 2013: SPARC T5: 16 cores, 8 threads/core, 3.6 GHz, etc.
  - 2016: SPARC M7: 4.13 GHz, 32 cores, 8 threads/core (256 threads!)
- In 2017, the 7th most powerful computer is SPARC based (and it is the first without GPU)

THE PAST

Alpha (DEC, 1992)

- 64 bit from the beginning
- Extremely innovative:
  - 21164: the first CPU with a big cache integrated with the CPU
  - 21264: the first CPU with both high frequency and out-of-order execution
  - 21364: the first CPU with integrated memory controller
  - 21464: supposed to be the first multi-thread CPU (but the project was stopped meanwhile)
- Extremely strong floating point unit
- 21264 @ 833 MHz > 3x Pentium III @ 1 GHz!
- Hand-made design
- Canceled when Compaq acquired DEC

PA-RISC (1986, HP)

- First 32, later 64 bit CPUs
- Extremely strong floating point unit
- PA-8600 @ 552 MHz > 2x Pentium III @ 1 GHz!
- Canceled when HP started to develop the Itanium processors with Intel

© Department of Networked Systems and Services

THE PAST (?)

IA-64 (Itanium)

- 1994, joint development of HP & Intel
- Huge interest from the press, very costly development
- First implementation: 2001, disappointing performance, sold only few thousand
- Supposed to be compatible with x86: succeeded, but it can reach only the level of a Pentium clocked at 100MHz...
- Problem: it needs a special compiler to utilize its abilities, they did not count with the difficulties of developing such compiler
- Still developed and manufactured, sold only 55.000 between 2001-2007
- Most big companies stopped supporting it
  - 2008: Microsoft
  - 2011: Oracle
- 2018: Intel announced to finish the production (till 2021)

COMPARISON

	x86	ARM	PowerPC	SPARC
Number of bits	64	32	64	64
Year introduced	1978	1983	1991	1985
Num of operands	2	3	3	3
Instruction style	Reg-mem	Reg-reg	Reg-reg	Reg-reg
CISC vs. RISC	CISC	RISC	RISC	RISC
Num of registers	8/16	16	32	32
Instruction coding	Variable (1-17)	Fixed (4)	Fixed (4 – com.)	Fixed (4)
Conditional instr.	Condition code	Condition code	Condition code	Condition code
Byte order	Little	Big	Big/Bi	Bi
Addressing modes	5	6	4	2
Branch predication	No	Yes	No	No

© Department of Networked Systems and Services

COMPARISON

	m68k	Alpha	PA-RISC	Itanium
Number of bits	32	64	64	64
Year introduced	1979	1992	1986	2001
Num of operands	2	3	3	3
Instruction style	Reg-mem	Reg-reg	Reg-reg	Reg-reg
CISC vs. RISC	CISC	RISC	RISC	EPIC
Num of registers	16	32	32	128
Instruction coding	Variable (2-22)	Fixed (4)	Fixed (4)	Fixed (16)
Conditional instr.	Condition code	Condition reg.	Compare & jump	?
Byte order	Big	Bi	Big	Bi
Addressing modes	9	1	5	?
Branch predication	No	No	No	Yes

© Department of Networked Systems and Services

DEPARTMENT OF NETWORKED SYSTEMS AND SERVICES

MŰEGYETEM 1782