von Neumann Architecture (machine)

1945 John von Neumann, (John Mauchly, Presper Eckert)

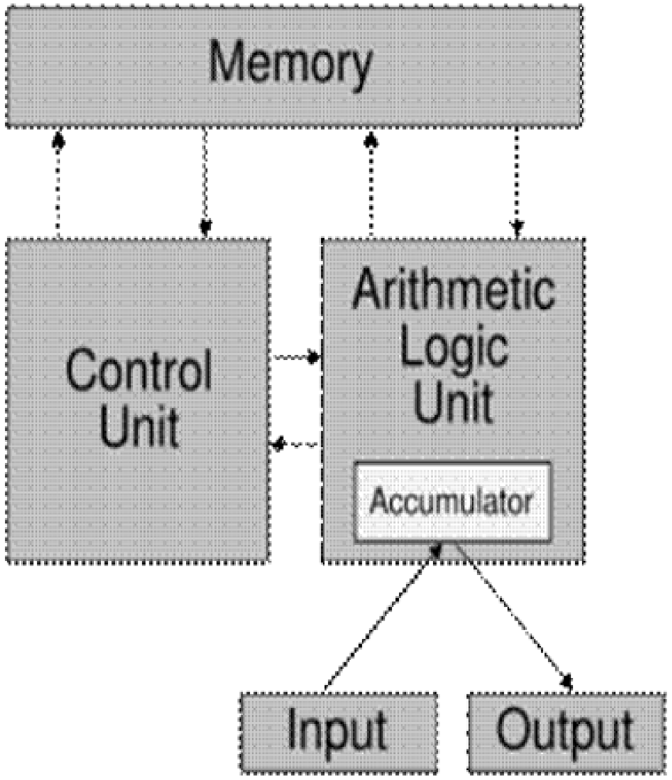
Model of the computer with the following main subsystems:

- Arithmetic Logic Unit (ALU)
- Control Unit (CU)

} Processor (CPU - Central Processing Unit)

- one common memory (and bus) for both:
  program instructions and data

- Input/Output (I/O) system



- many modern computers and CPUs are more or less based on the Von Neumann Architecture:
IBM PC, Motorola MC6800…

von Neumann Architecture

- **program instructions and data are stored in one common memory** as numbers (in binary form)

- instructions and data are transferred between memory and CPU through **common shared bus**

- memory is divided into storage units of fixed size: memory cells.
- each memory cell has an unique address
- memory cell is minimum accessible unit of memory

- CPU executes program instructions sequentially

The Simplest Memory Organization

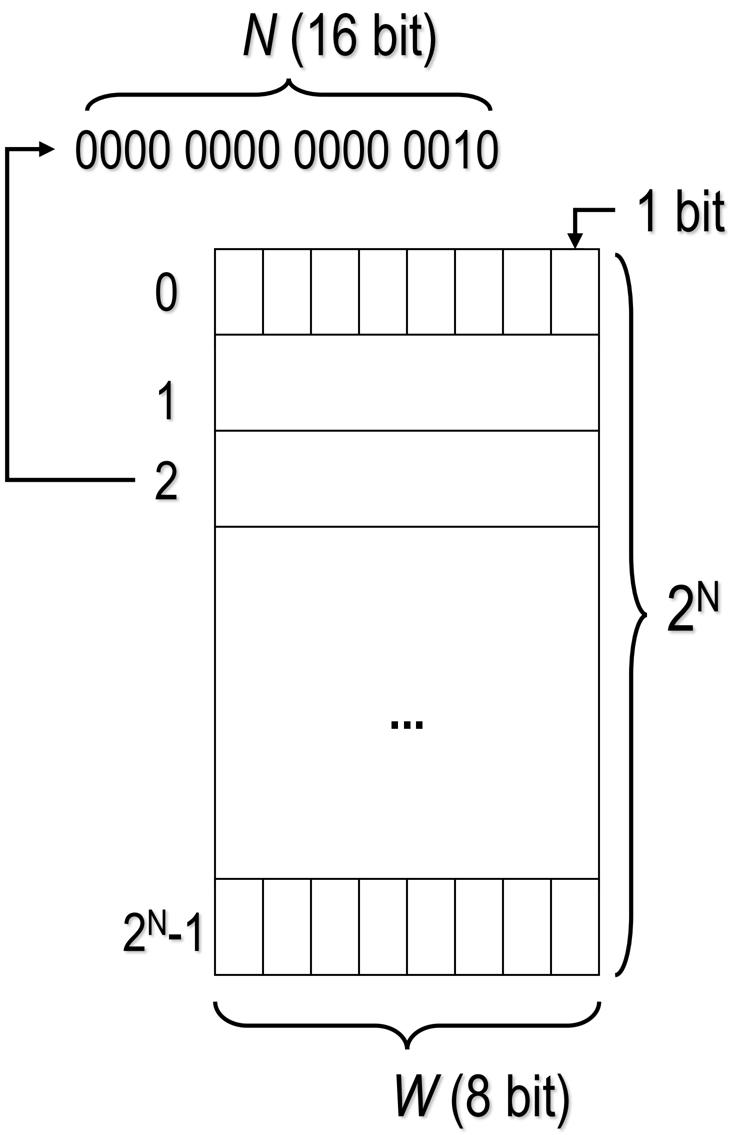
- memory width (*W*) (databus width)
  - number of bits in each memory cell
    here: 8 bits (1 byte)

- address width (*N*) (address bus width)
  - number of bits used to represent
    memory address

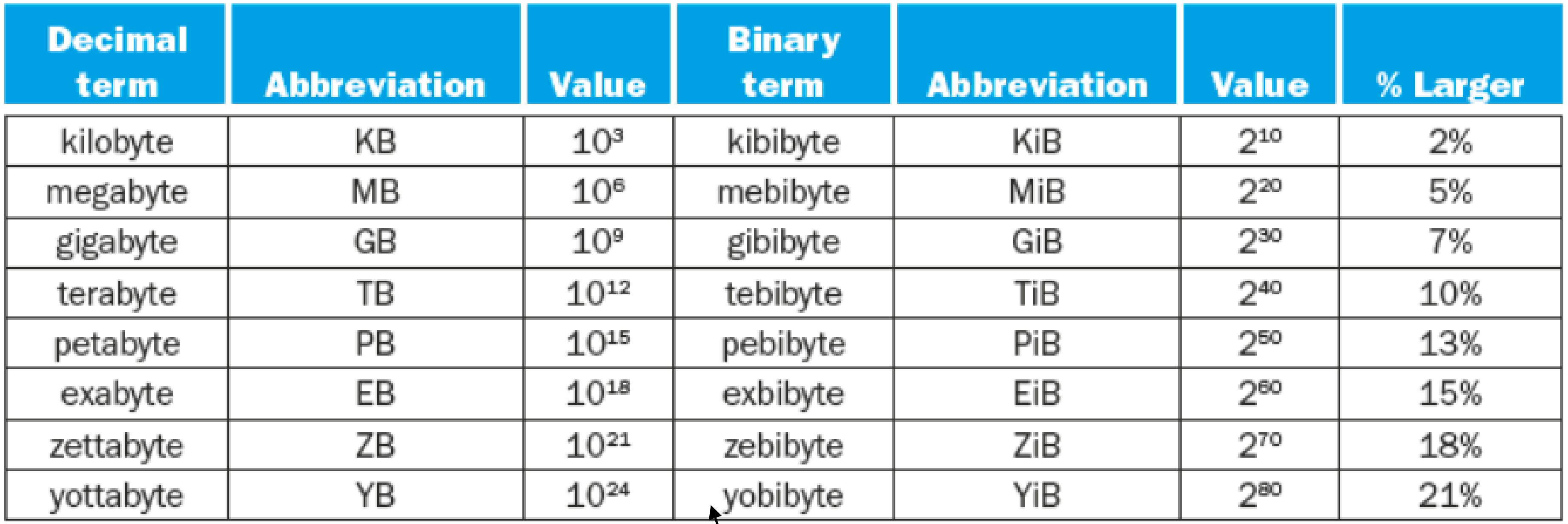
Address space = max. memory size (2^N)
(addressable from 0 to 2^N-1)

E.g. 8bit 6800, 6502, Z80 CPUs have:
8bit data bus
16bit address bus -> 64 KB address space

Intel 8088
8bit external (16bit internal) data bus
20bit addresses -> 1 MB address space



Binary prefixes – IEC 1998 (International Electrotechnical Commission)



Decimal term	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

1 kV =1000 V
1 MΩ =1000 kΩ = 1000000 Ω

1 nF = 1/1000 µF
…

1 KiB = 1024 Bytes
1 MiB = 1024 KiB = 1048576 Bytes

similarly: kilobits and kibibits…

Instruction Processing

- CPU is a kind of sequential circuit:

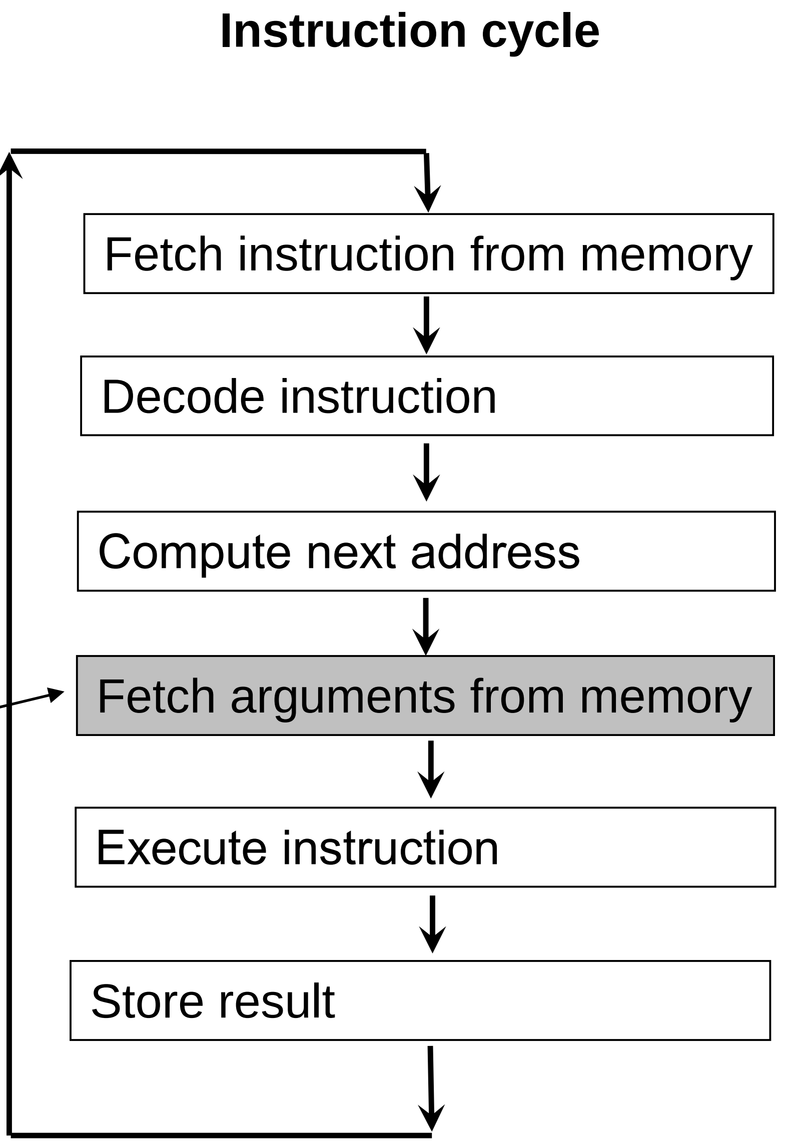
- a finite state machine:

i.e. executes instructions
sequentially and repeatedly,
clocked by the pulses of a clock signal

- not all phases are needed
  by every instruction

- each phase of instruction cycle is called
  machine cycle.

- machine cycles may take variable
  number of clock ticks
  (sometimes 1 clock pulse per 1 machine cycle)



von Neumann Architecture

von Neumann bottleneck:

- only one bus, used for both data transfers and instruction fetches:
- data transfers and instruction fetches must be scheduled
  (they can not be performed at the same time)
- throughput (data transfer rate) between the CPU and memory is limited

program code stored in Random Access Memory can be modified during runtime!

- selfmodification
- accidental modification, due to errors
- malicious modification

Memory protection is needed…

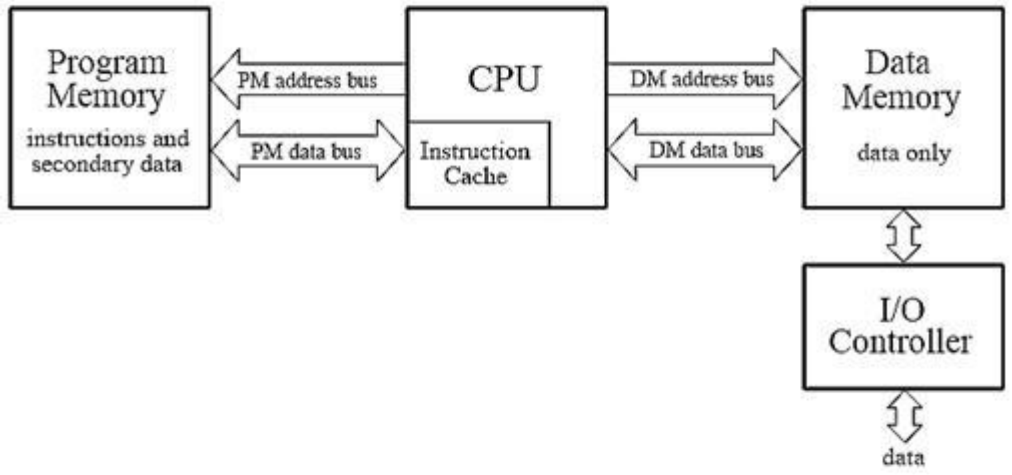
Harvard Architecture

• (at least…) **two separated memory systems for instructions and data**
• connected to the CPU through **separated buses**

• instructions and data can be fetched and transferred simultaneously on both busses

- greater throughput (data transfer rate)

• address spaces, data widths, implementation technologies of these memories can differ

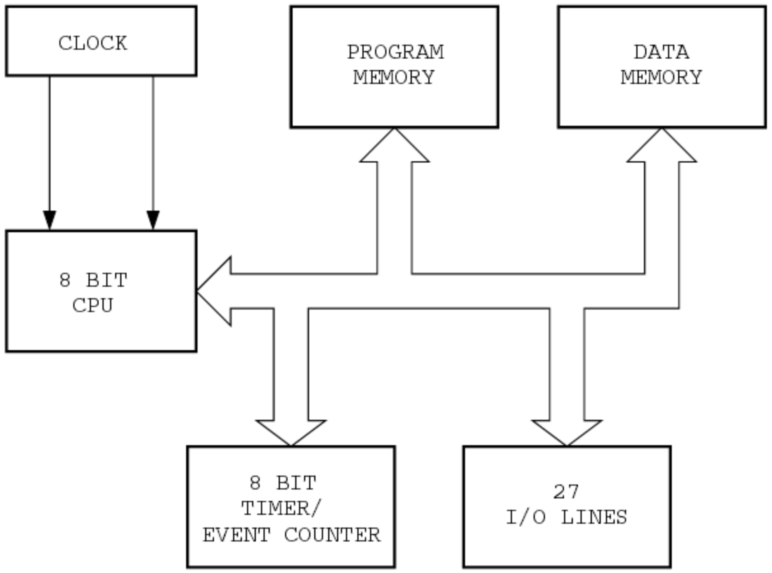


Modified Harvard Architecture (1)

- **separate memory blocks** of different type with **one shared bus**

  simple 8bit microcontrollers (single-chip microcomputers like 8051, AVRs)
  Program: Read-Only-Memory (Non-volatile, Flash, EPROM)
  Data: Random-Access-Memory (Read-Write)

- program memory can store constant data (fonts, sounds, text strings)
- special instructions allow to load them directly to the CPU core



Modified Harvard Architecture (2)

Memory hierarchy in modern personal computer

- main operational memory – von Neumann architecture

- level 1 cache – Harvard architecture

    - concurrent access to the instructions and data may reduce stalls and structural hazards in pipelined processors

Computer Architecture

Intuitive definition:

Hardware organization. Configuration of the basic blocks (CPU, memory, I/O…) and connections between them…

and more formal:

Functional operation of the individual hardware units within a computer system, and the flow of information and control among them.

(from Richard C. Dorf Computers, Software Engineering, and Digital Devices)

But… is this really all?

Computer Architecture

Any digital computer (with CPU, memory and I/O) can (given enough time) complete any algorithm*

like Infinite Monkey Theorem…

*taken from:
Using the Intel MCS-51 Boolean processing capabilities, Intel Corporation, 1980

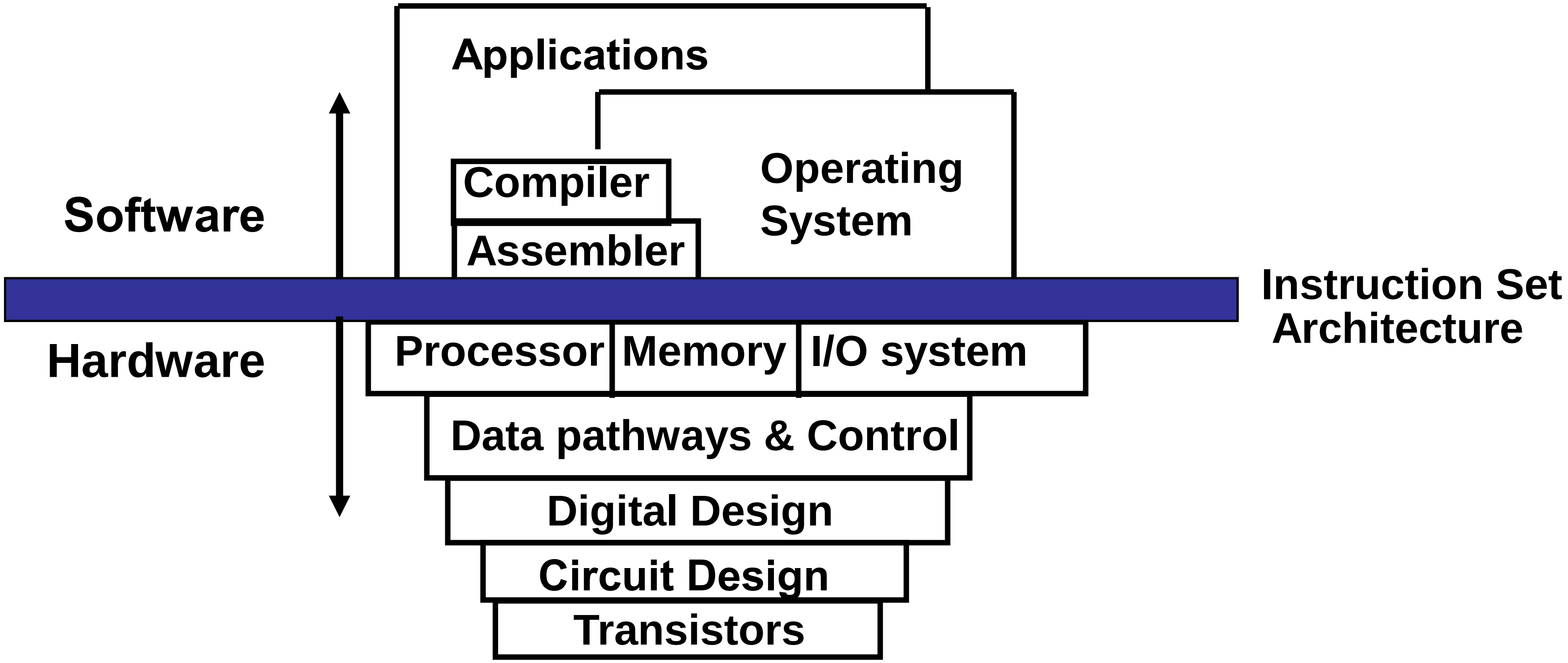
Computer Architecture

ISA – Instruction Set Architecture – programmer's model

i.e. how computer system looks to programmer:

- number and type of registers
- instruction formats, operation code set
- specifies data types and data structures and storage elements
- addressing modes and accessing data items
- I/O handling

Computer Architecture = ISA + Hardware Organization



Instruction Set Architecture

two major concepts:

CISC – Complex Instruction Set Computer

RISC – Reduced Instruction Set Computer

CISC – Complex Instruction Set Computer

An older approach:

- implements a large instruction set (100…>>1000), from simple ones to very complex
- usually microcoded
- instructions of varying length (form one byte to dozens of bytes)
- complex coding schemes
- small number of general purpose registers (e.g. 4…16)
- arguments can be directly fetched from main memory
- execution times differ, depending on instruction type,
  from several clock cycles to several dozen clock cycles

„Classic" CPUs designed in 70s: Intel 8080, 8085, 8086 (x86) Z80,
Motorola 6800, microcontrollers: 8048, 8051, 6805 etc

CISC – Complex Instruction Set Computer

Pros:

- various complex instructions and addressing modes make the assembly programs shorter and easier to write by human

- high density, compact code – important in embedded systems (e.g. microcontrollers with very small program memory 1-4KB)

Cons:

- complicated digital design, hard to validate and debug, high power consumption (x86)

- many of the implemented features were used rarely or not used at all…

- instructions of different sizes, execution times and complexity create problems with:

    - pipelined processing
    - instruction level parallelism
    - automatic code optimization (by compiler)

CISC – Complex Instruction Set Computer

complex instruction examples (x86):

- **copy the block of data**

```
    MOV  $source_address , %esi
    MOV  $destination_address, %edi
    MOV  $number_of_bytes, %ecx
    CLD                              ;clear direction flag
    REPNZ MOVSB                      ;move string of bytes until ecx
                            ;reaches zero
```

- **for loop**

```
    MOV  $10 , %ecx                  ;number of iterations
for_label:

   ;some  code

    LOOP for_label          ;decrement ecx, then jump to label if ecx!=0
```

- such instructions are „hard-wired” with certain registers, (e.g. LOOP with ecx)
- were designed over 40 years ago, now - are not optimized for pipeline execution

x86 architecture - instruction encoding: a real mess! (only 32bit version is shown)

reg	w = 0	w = 1		r/m	mod = 0		mod = 1		mod = 2		mod = 3
		16b	32b		16b	32b	16b	32b	16b	32b	
0	AL	AX	EAX	0	addr=BX+SI	=EAX	*same*	*same*	*same*	*same*	*same*
1	CL	CX	ECX	1	addr=BX+DI	=ECX	*addr as*	*addr as*	*addr as*	*addr as*	*as*
2	DL	DX	EDX	2	addr=BP+SI	=EDX	*mod=0*	*mod=0*	*mod=0*	*mod=0*	*reg*
3	BL	BX	EBX	3	addr=BP+SI	=EBX	*+ disp8*	*+ disp8*	*+ disp16*	*+ disp32*	*field*
4	AH	SP	ESP	4	addr=SI	=*(sib)*	SI+disp8	*(sib)*+disp8	SI+disp8	*(sib)*+disp32	"
5	CH	BP	EBP	5	addr=DI	=disp32	DI+disp8	EBP+disp8	DI+disp16	EBP+disp32	"
6	DH	SI	ESI	6	addr=disp16	=ESI	BP+disp8	ESI+disp8	BP+disp16	ESI+disp32	"
7	BH	DI	EDI	7	addr=BX	=EDI	BX+disp8	EDI+disp8	BX+disp16	EDI+disp32	"

a. JE EIP + displacement

4	4	8
JE	Condi-tion	Displacement

b. CALL

8	32
CALL	Offset

c. MOV EBX, [EDI + 45]

6	1	1	8	8
MOV	d	w	r/m Postbyte	Displacement

d. PUSH ESI

5	3
PUSH	Reg

e. ADD EAX, #6765

4	3	1	32
ADD	Reg	w	Immediate

f. TEST EDX, #42

7	1	8	32
TEST	w	Postbyte	Immediate

- instruction-dependent, various encoding schemes
- instructions and their arguments are encoded as a sequence of bits of different length
- certain instructions have „strange” lengths: not a common powers of two but odd: 5-7 bytes

RISC – Reduced Instruction Set Computer

- instruction set is small and as simple as possible,
  with as few as 30–50 necessary instructions
- (almost…) all instructions are executed in a single cycle
- all instructions have the same size and a fixed format (encoding scheme)
  thus, are very simple to decode
- large number of general purpose registers

Load Store concept:

- The processor access data from external memory with explicit Instructions: Load and Store.
- All other operations, such as adds, sub-
  tracts, and logical operations, use only fast registers built-in CPU

Rapid development in the 80s: MIPS, ARM, Sun SPARC, IBM PowerPC…

RISC – Reduced Instruction Set Computer

PROS and CONS

- RISC usually requires more (simple) instructions to solve a problem
- but this is compensated for by the fact that each instruction is executed much faster so the overall running time is less.
- large number of the short and simple instructions allow effective compiler optimization
- hardware complexity is reduced, thus RISC machines are easier to validate,
- properly written code has statistically smaller number of memory accesses (only load/store instructions)
- compact, uniform instructions - facilitate pipelining
- more instructions in program: larger memory consumption, bigger executable files

Instructions are typically 4 (or 2) bytes long
thus, have enough space to encode 3 operands:

```
        ADD R0,R1,R2;     R0= R1+R2
```

The world has become too complex to talk only about RISC vs. CISC!

- in the modern x86 (>= Pentium, Pentium Pro) classic CISC instructions are internally „translated” to the sequences of RISC-like microoperations…

- ARM (RISC) processors can conditionally execute each instruction:

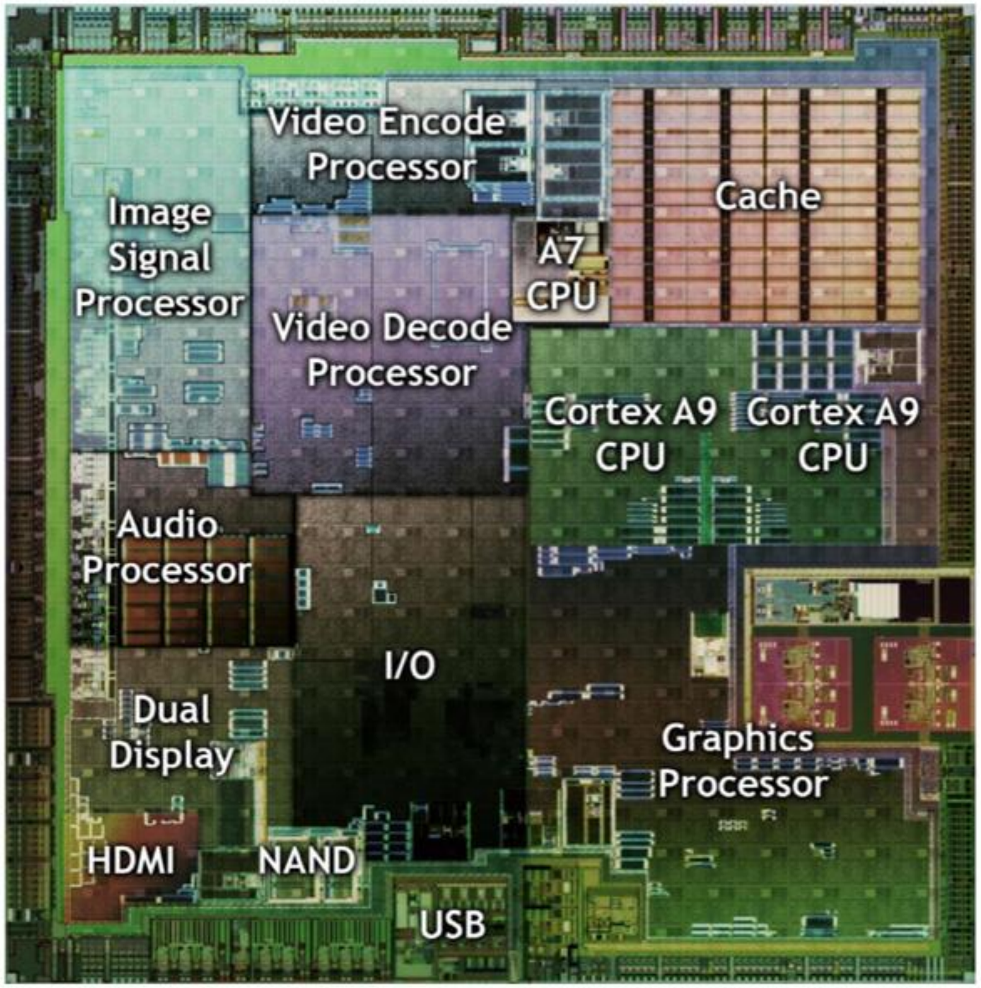
ADDEQ R3,R2,R1,LSL #2

if equal (zero_flag==1) then R3=R2+4*R1

and are equipped with sets of:

- floating point
- vector
- digital signal processing

instructions.



MIPS (Microprocessor without Interlocked Pipe Stages)
ARM (Advanced RISC Machine) - most common in portable devices

a well designed, simple instruction encoding schemes:

all instructions are
32/16 bit long
(powers of two -
easy to align and access
in the memory space)

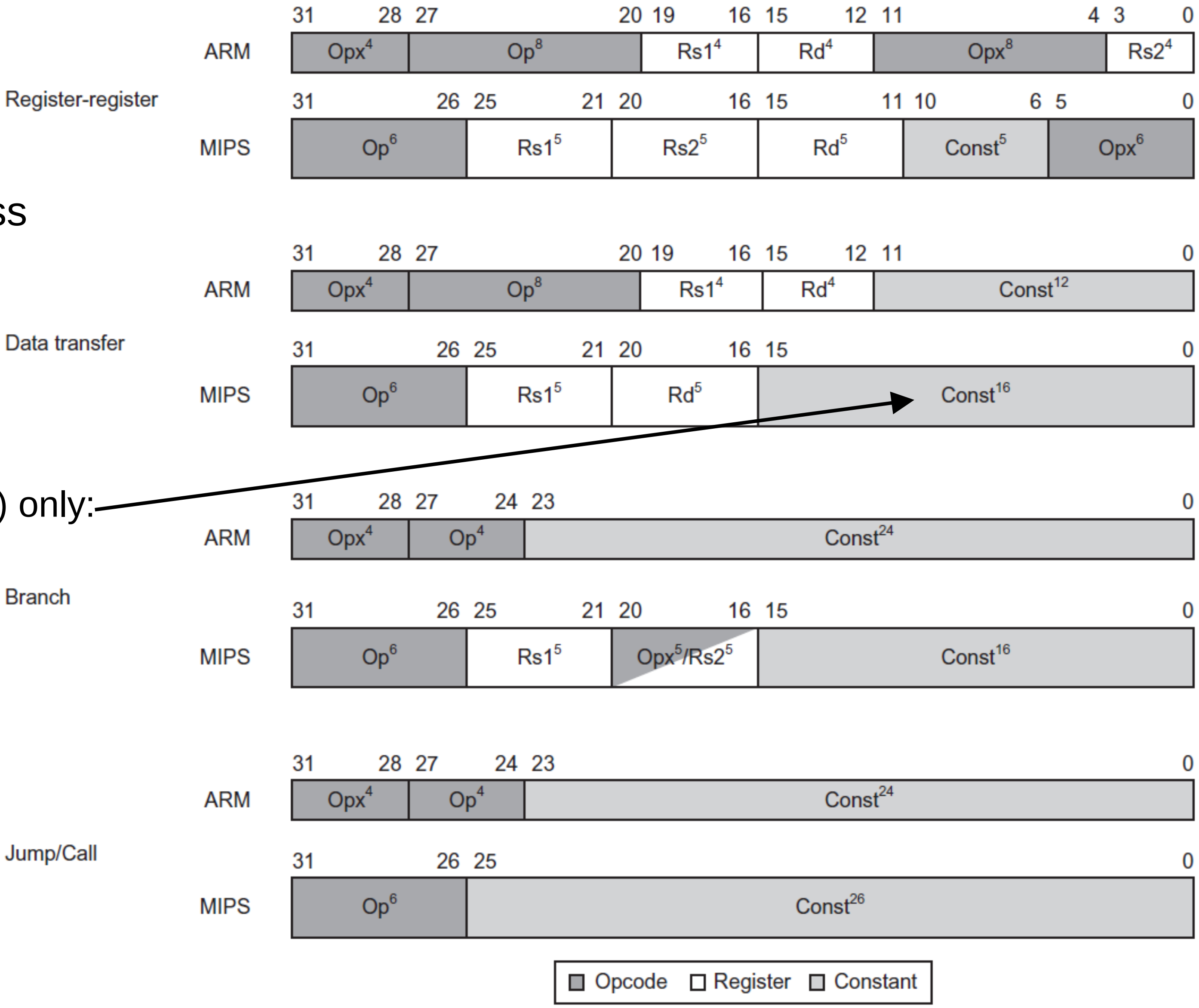
16bit constant (immediate) only:

32bit constant is loaded
in two steps:
1. Load Upper Immediate)

lui $s1, 35

2. Insert the lower half:

ori $s1, $s1, 46


Register-register
ARM
31 28 27 20 19 16 15 12 11 4 3 0
Opx4 | Op8 | Rs14 | Rd4 | Opx8 | Rs24
MIPS
31 26 25 21 20 16 15 11 10 6 5 0
Op6 | Rs15 | Rs25 | Rd5 | Const5 | Opx6
Data transfer
ARM
31 28 27 20 19 16 15 12 11 0
Opx4 | Op8 | Rs14 | Rd4 | Const12
MIPS
31 26 25 21 20 16 15 0
Op6 | Rs15 | Rd5 | Const16
Branch
ARM
31 28 27 24 23 0
Opx4 | Op4 | Const24
MIPS
31 26 25 21 20 16 15 0
Op6 | Rs15 | Opx5/Rs25 | Const16
Jump/Call
ARM
31 28 27 24 23 0
Opx4 | Op4 | Const24
MIPS
31 26 25 0
Op6 | Const26
Opcode
Register
Constant