Chapter15 Flipflop-Circuit

1. Counter
2. Register

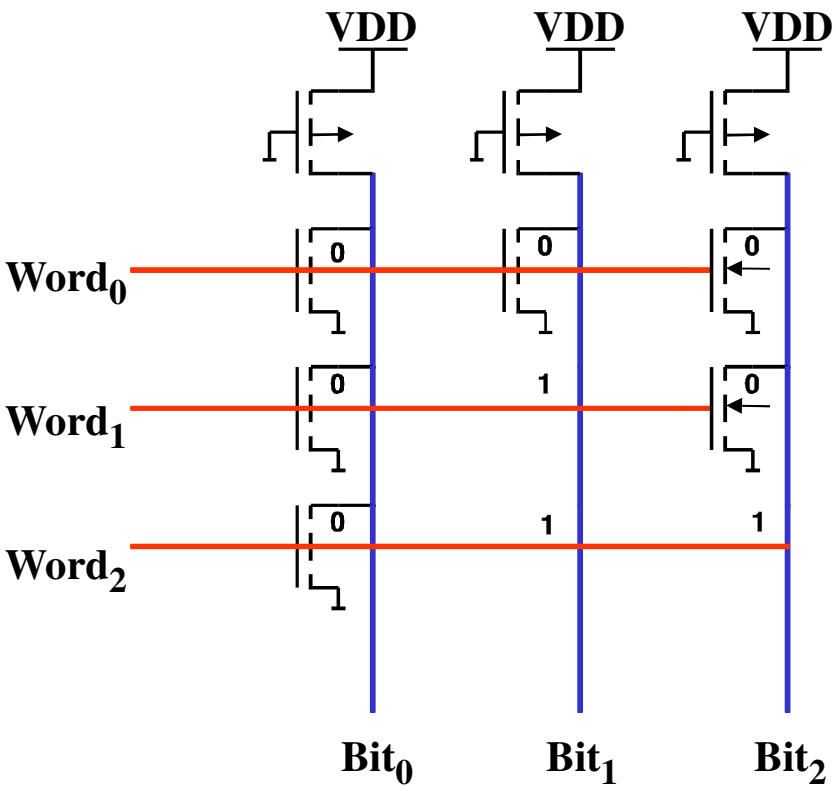
Chapter 16 Memory

1. RAM - Random Access Memory
   Static RAM (SRAM)
   dynamic RAM (DRAM)
2. NVM (non-volatile memory)
3. ROM – Read-Only Memory

Standard Logic Devices

1. PLA - Programmable Logic-Arrays
2. FPGA

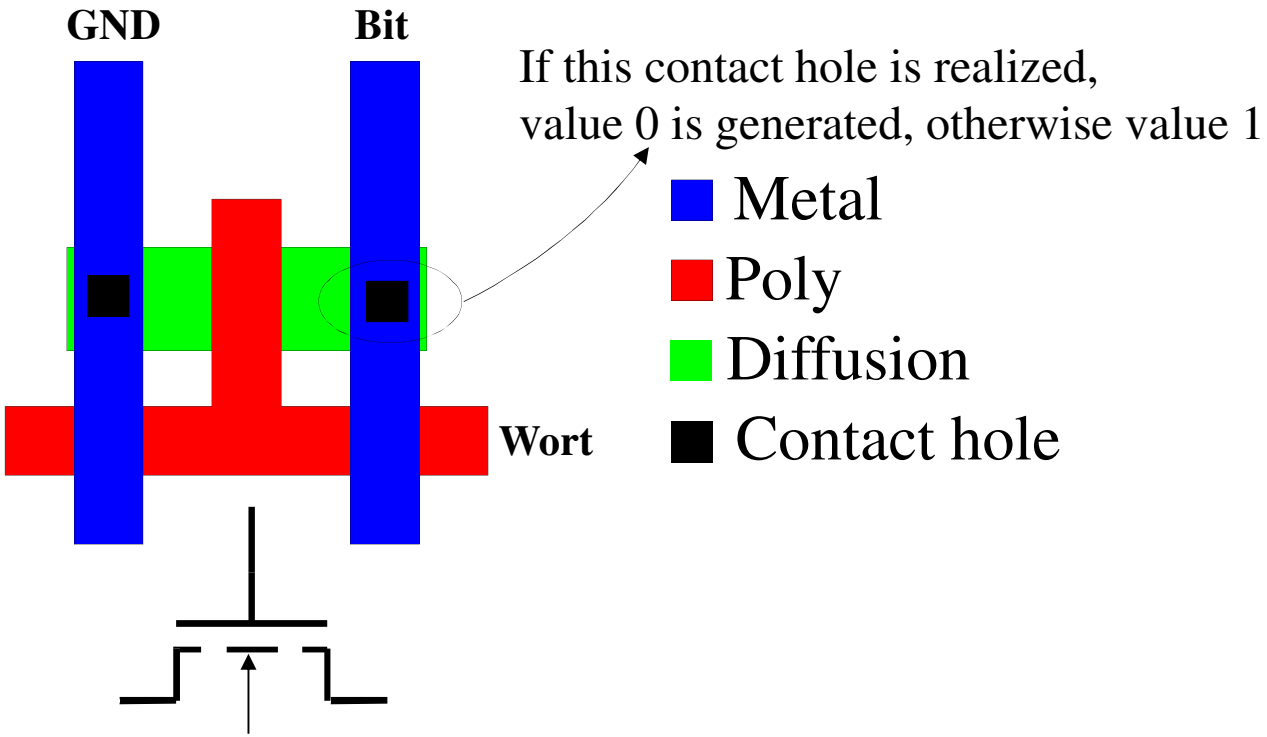
ROM-Read-Only Memory



Word 0	0	0	0
Word 1	0	1	0
Word 2	0	1	1

ROM-Memory Cell Array (NOR Structure)

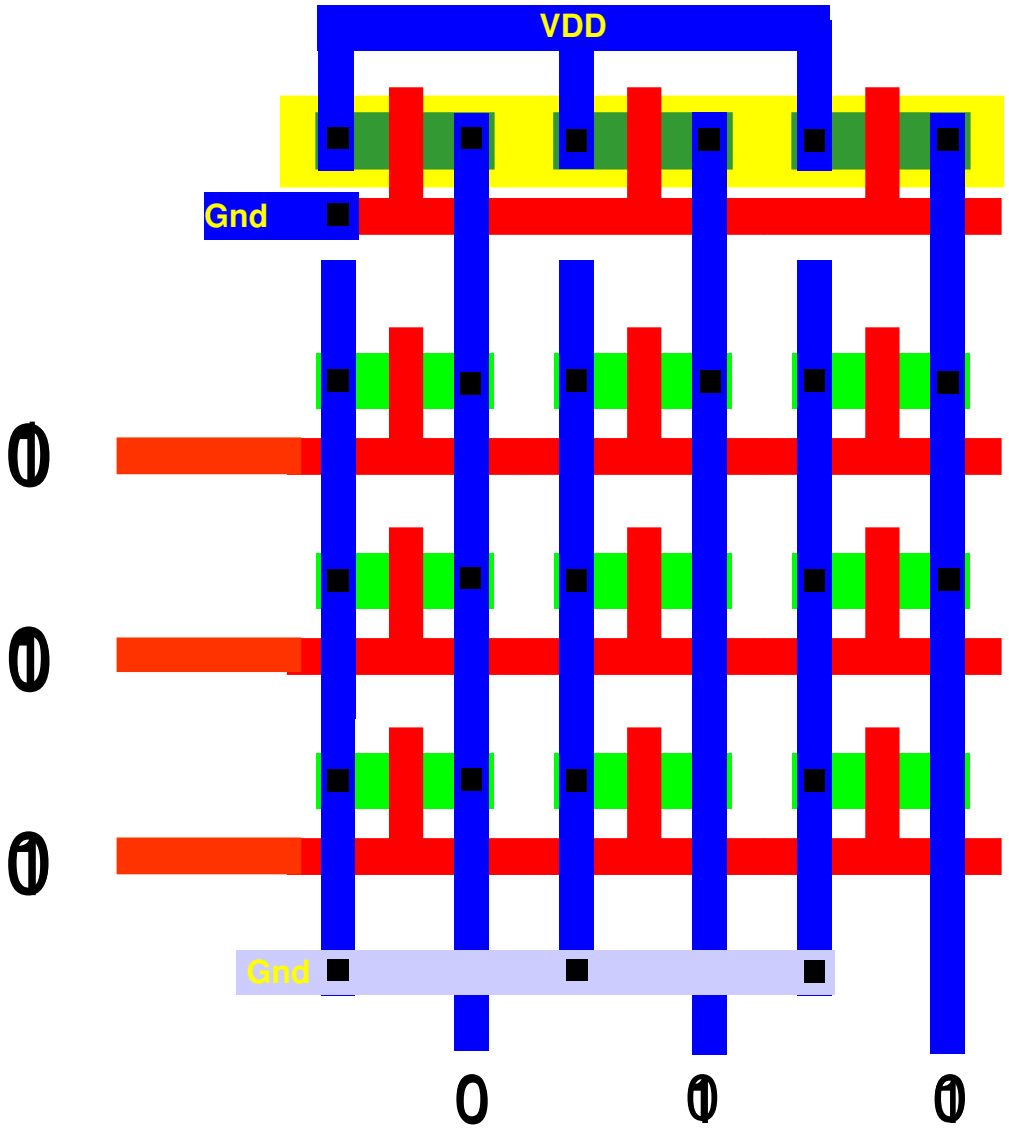
Memory Transistor of a ROM-Cell



Specifics: Programming on a mask (contact hole / via)

+ high storage density
- inflexible

3 X 3 ROM Layout



PMOS-Current source

Word 0	0	0	0
Word 1	0	1	0
Word 2	0	1	1

With the contact hole mask the ROM is programmed

Chapter15 Flipflop-Circuit

1. Counter
2. Register

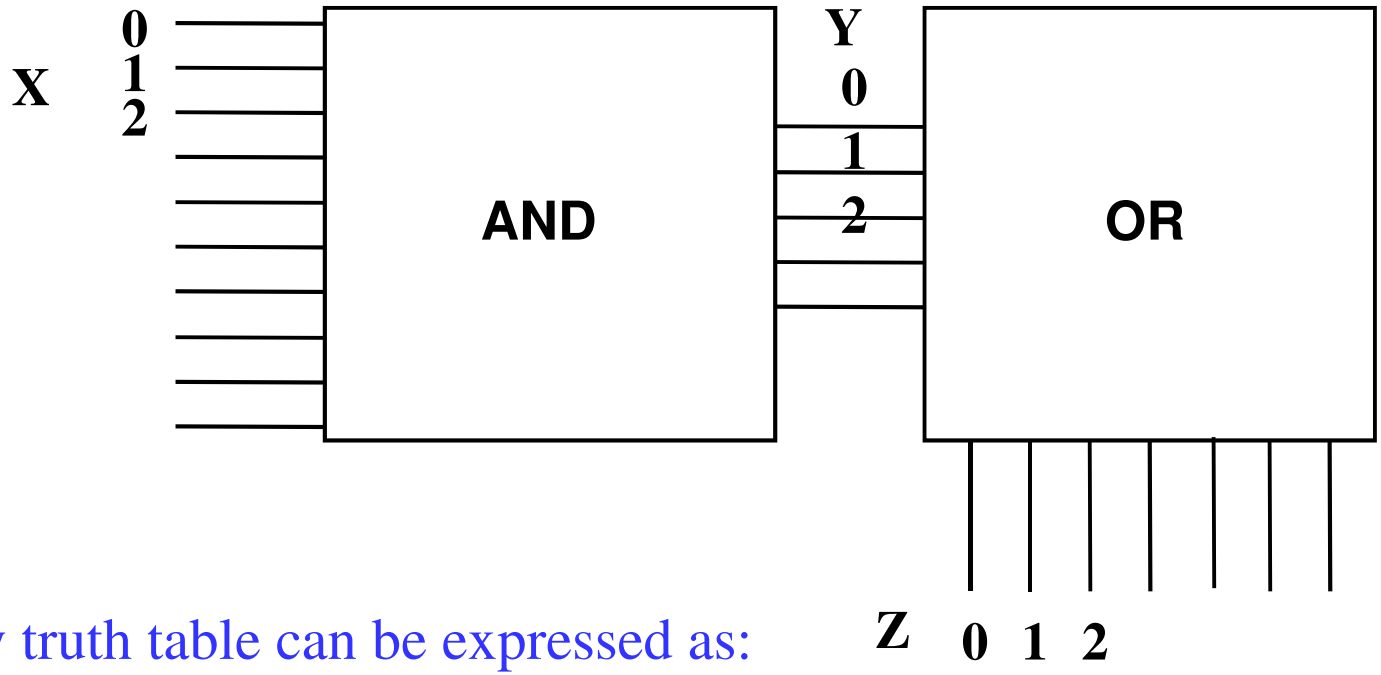
Chapter 16 Memory

1. RAM - Random Access Memory
   Static RAM (SRAM)
   dynamic RAM (DRAM)
2. NVM (non-volatile memory)
3. ROM – Read-Only Memory

Standard Logic Devices

1. PLA - Programmable Logic-Arrays
2. FPGA

PLA - Programmable Logic-Arrays



Any truth table can be expressed as:

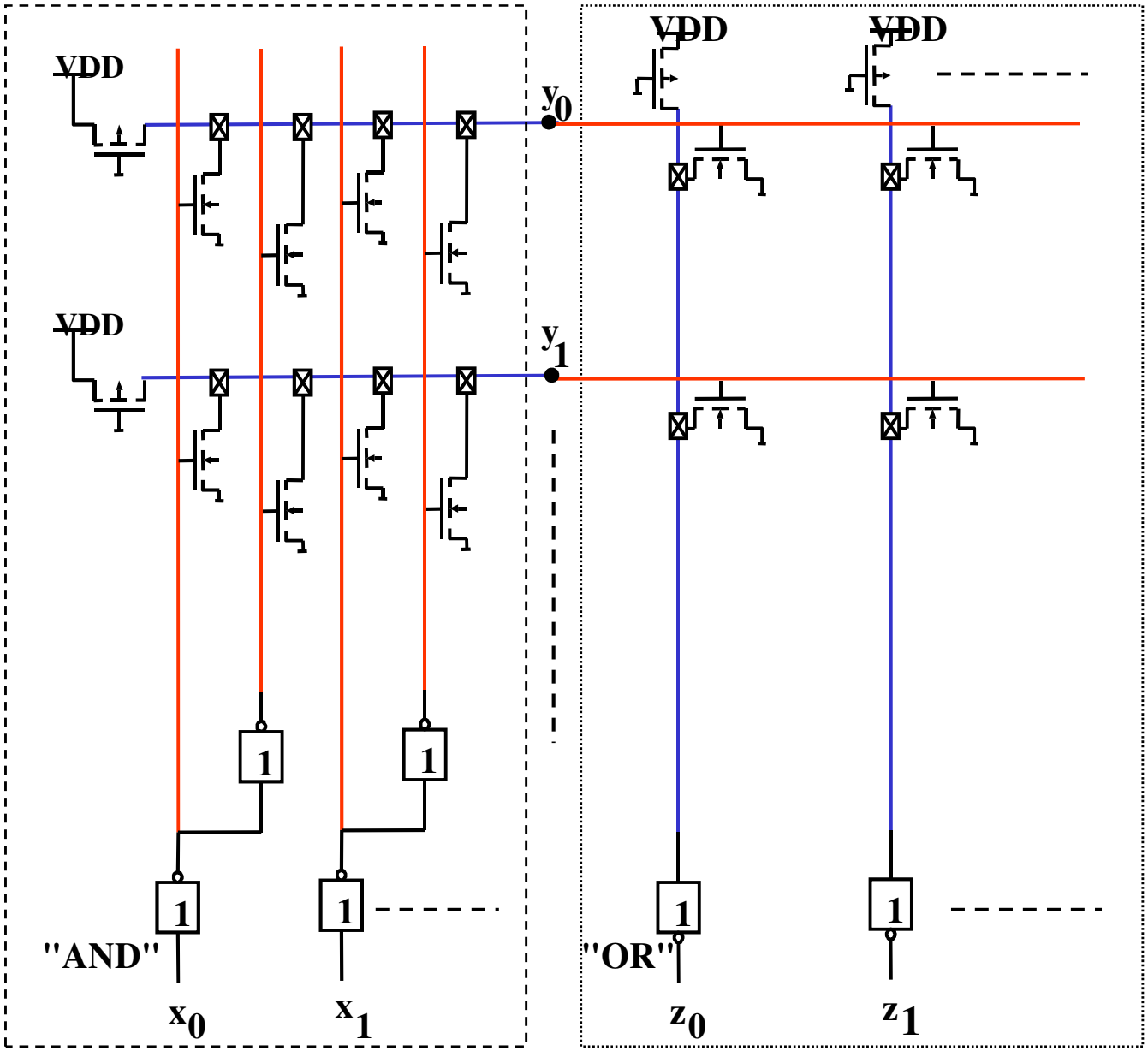
$$Z_0 = X_0 \square X_2 + X_0 \square \overline{X_1} \square X_3 + \overline{X_2} \square \overline{X_4} \square X_5$$

$$Y_{0\ldots m} = AND(X_{0\ldots n})$$

$$Z_{0\ldots l} = OR(Y_{0\ldots m})$$

Realisierung: $$Z_0 = \overline{\left(\overline{X_0} + \overline{X_2}\right)} + \overline{\left(\overline{X_0} + X_1 + \overline{X_3}\right)} + \overline{\left(X_2 + X_4 + \overline{X_5}\right)}$$

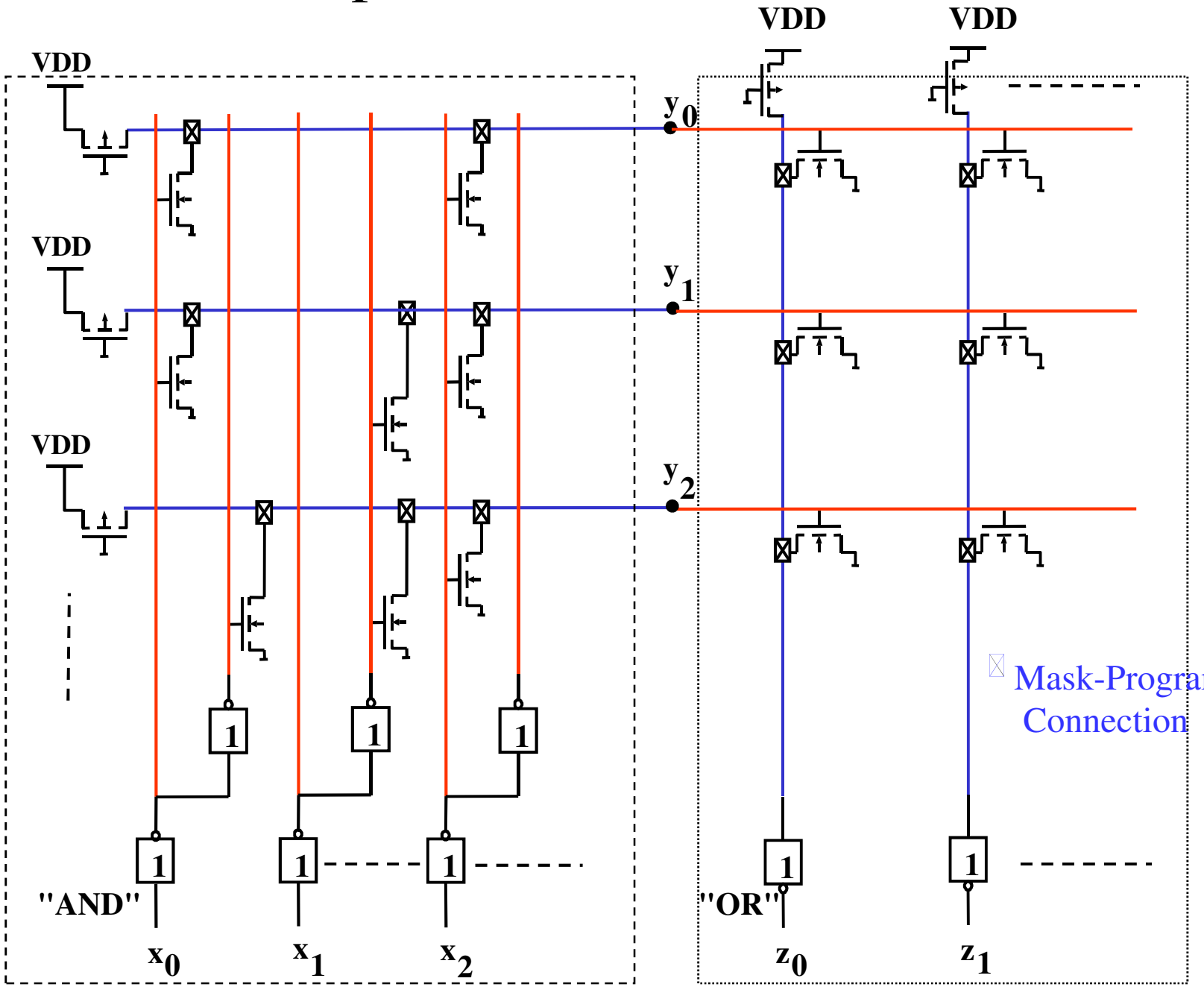
PLA - Array



Mask-Programmed Connection

PLA - Implementation

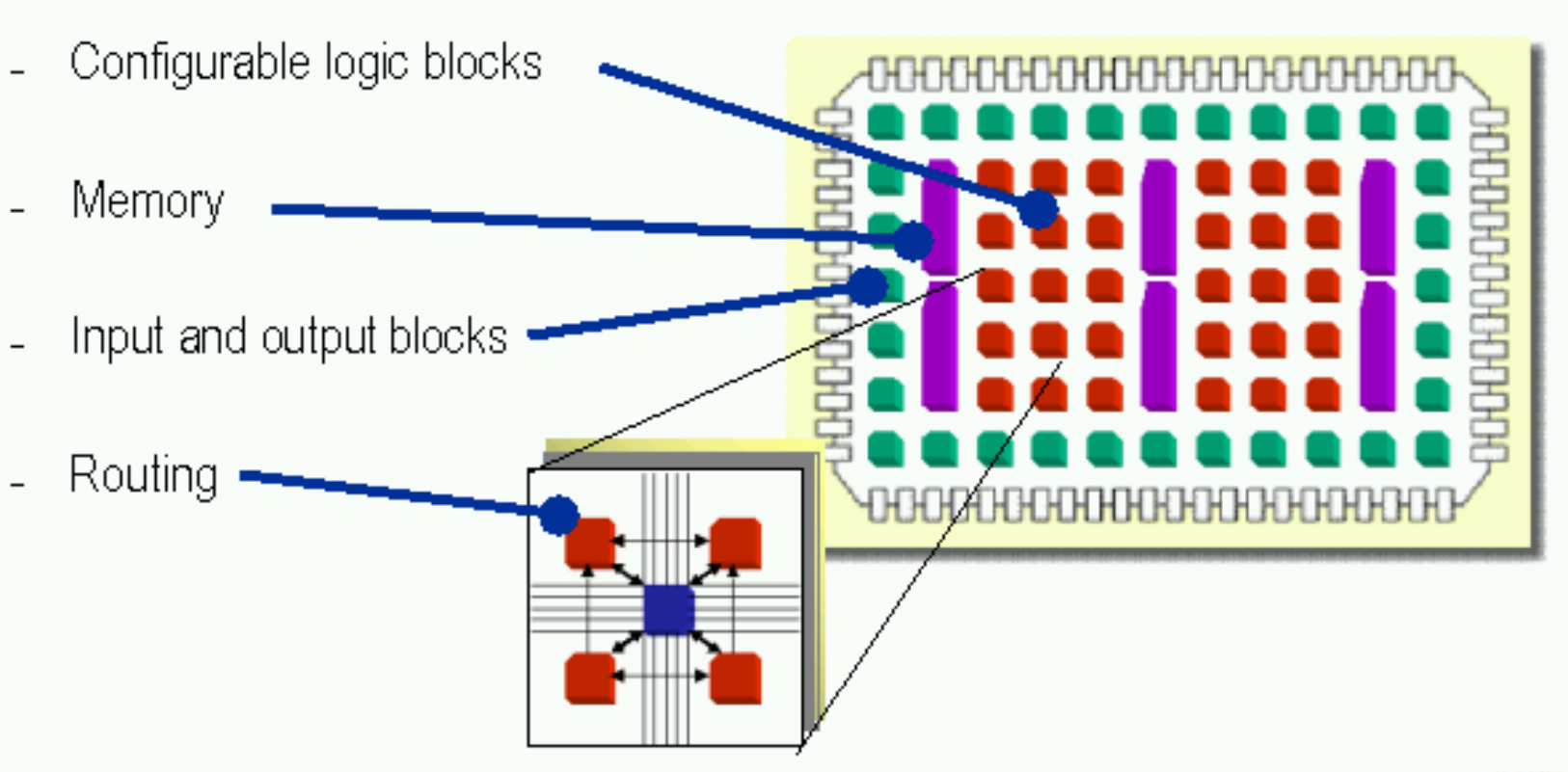
$$Z_0 = \overline{\left(\overline{X_0} + \overline{X_2}\right)} + \overline{\left(\overline{X_0} + X_1 + \overline{X_2}\right)} + \overline{\left(X_0 + X_1 + \overline{X_2}\right)}$$



FPGA (Field Programmable Gate Array)

- Functionality of the device is programmable in the field
- Hardware blocks (Register, Memories, Arithmetic Logic) are fixed
- Wiring and combinatorial functions are programmable
- Reduced development times
- Bugfixes are possible
- Higher costs per unit
- Lower clock frequency

Basis Elements of FPGA



I/O: Reading the Programs via external Bus or Flash

Memory: Control of wiring,

Lookup-Tables (Combinatorial logics)

Application areas of FPGA

Between ASIC (application specific IC) and Universal-Processor

Prototypes (also for later ASIC production)

Specific applications in Medicine, Industry, etc. (small series)

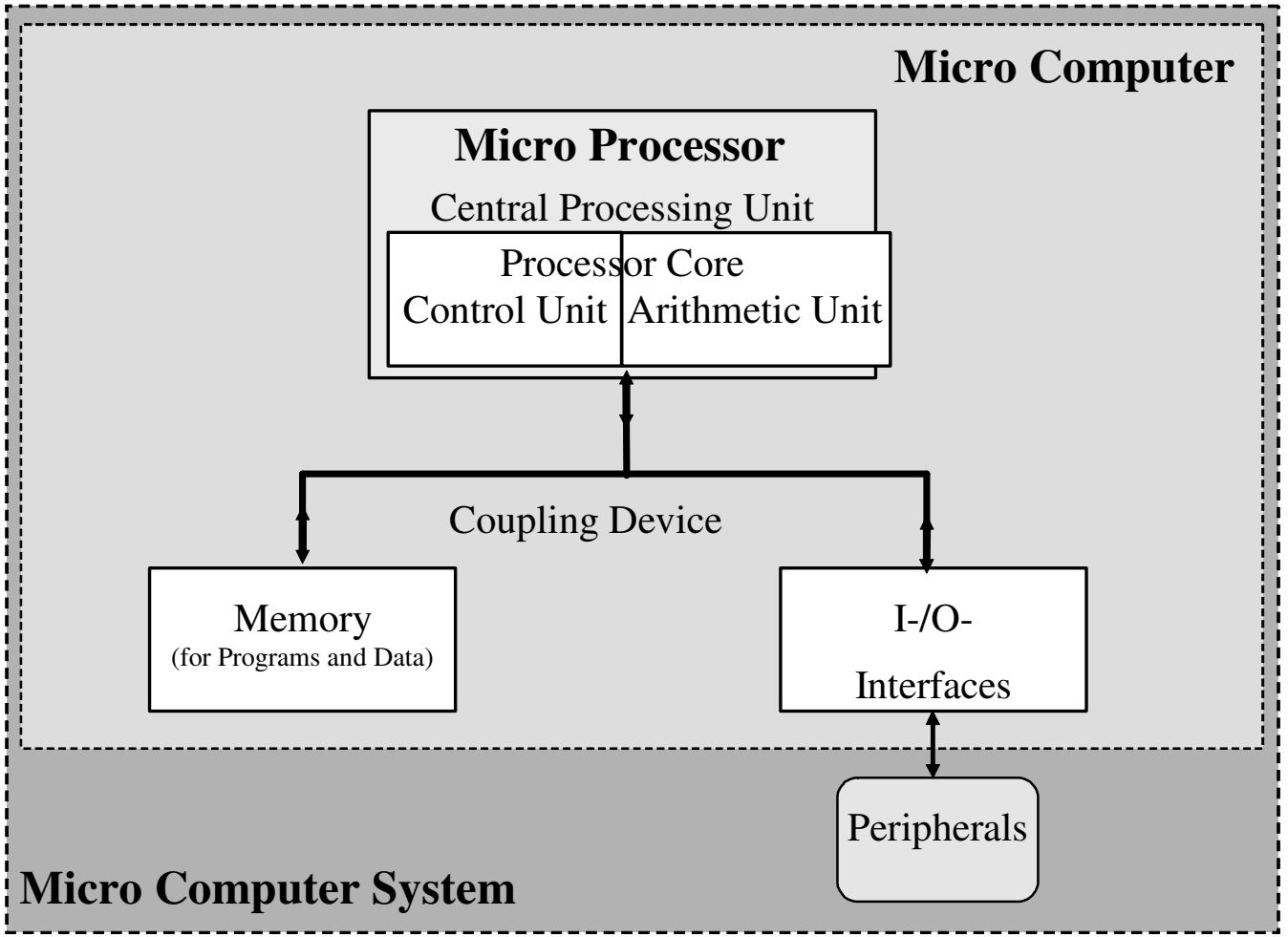
Implementation in FPGA

Synthesis of HDL (Hardware description language) in FPGA

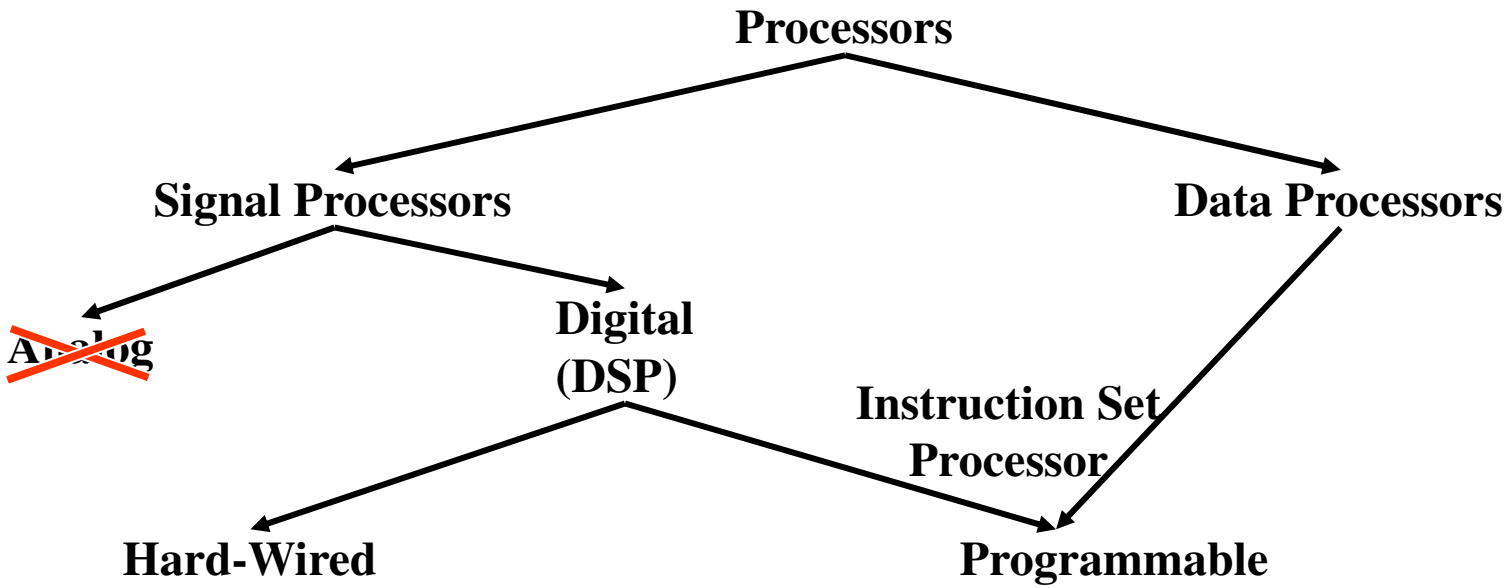
Major Manufacturer: Xilinx, Altera

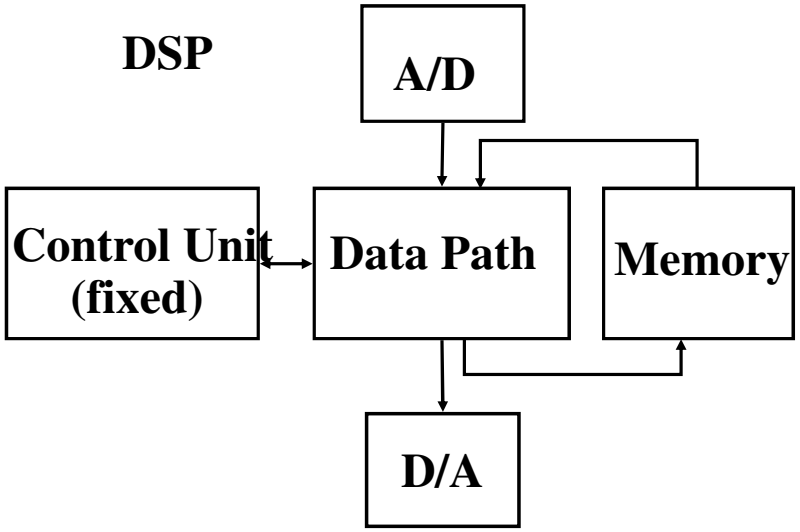
Processor

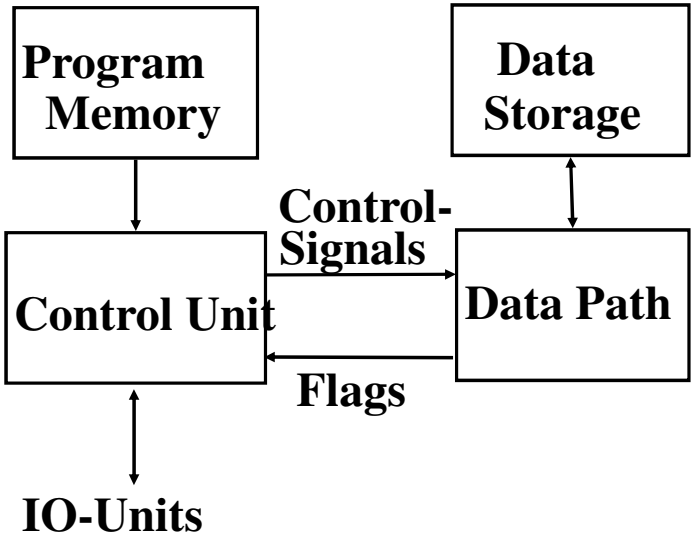
- Applications
- Structure
- Addressing
- Instructions



Classification of Processors







Processor Applications

General Purpose - high performance

- Pentiums, Alpha's, SPARC, Power PC
- Used for general purpose software
- Heavy weight OS - UNIX, NT
- Workstations, PC's

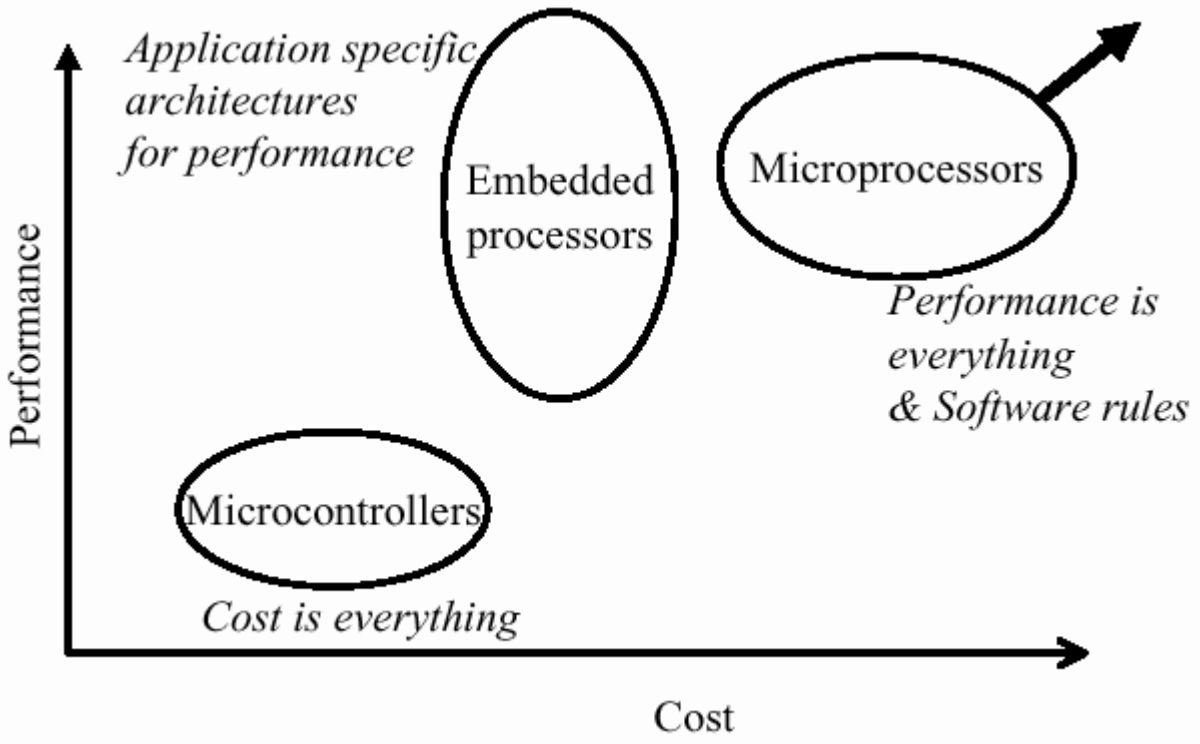
Embedded processors and processor cores

- ARM, 486SX, Hitachi SH7000, NEC V800
- Single program
- Lightweight, often realtime OS
- DSP support
- Cellular phones, consumer electronics (e. g. CD players)

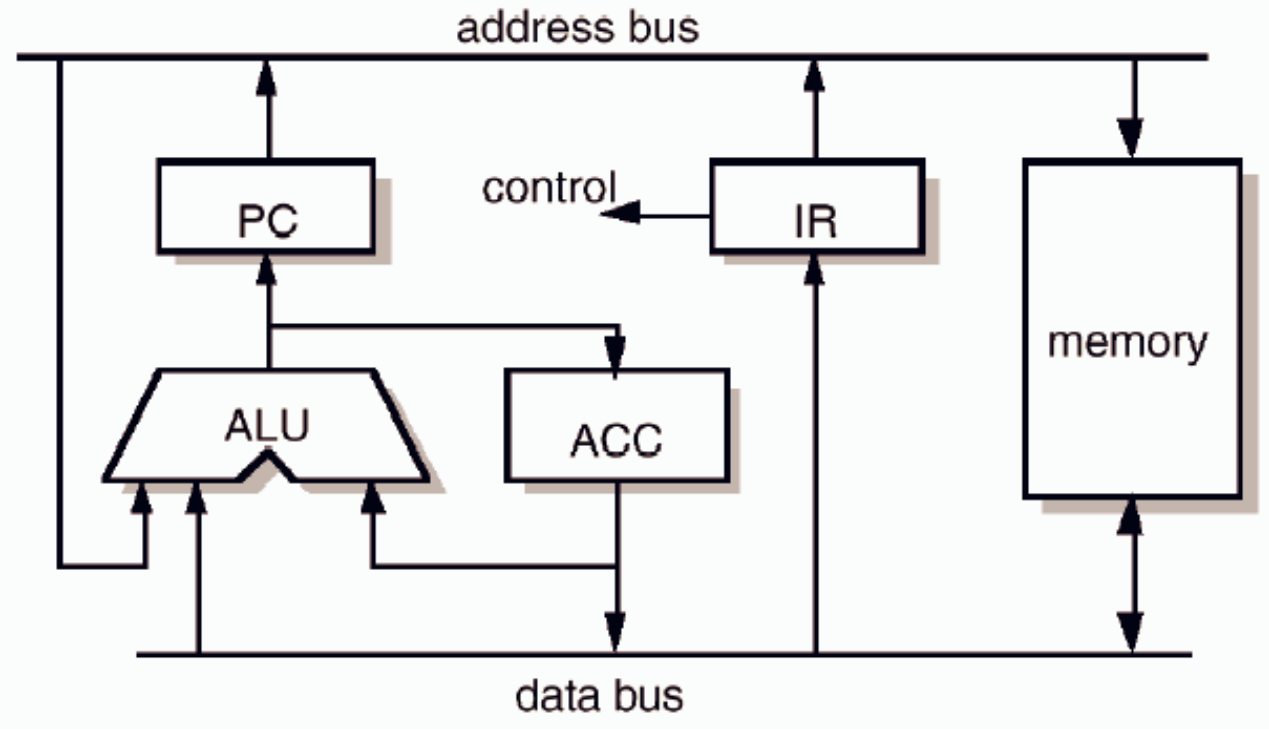
Microcontrollers

- Highest volume processors by far, extremely cost sensitive
- Small word size - 8 bit common
- Automobiles, toasters, thermostats, ...

Processor Applications



A Simple Processor



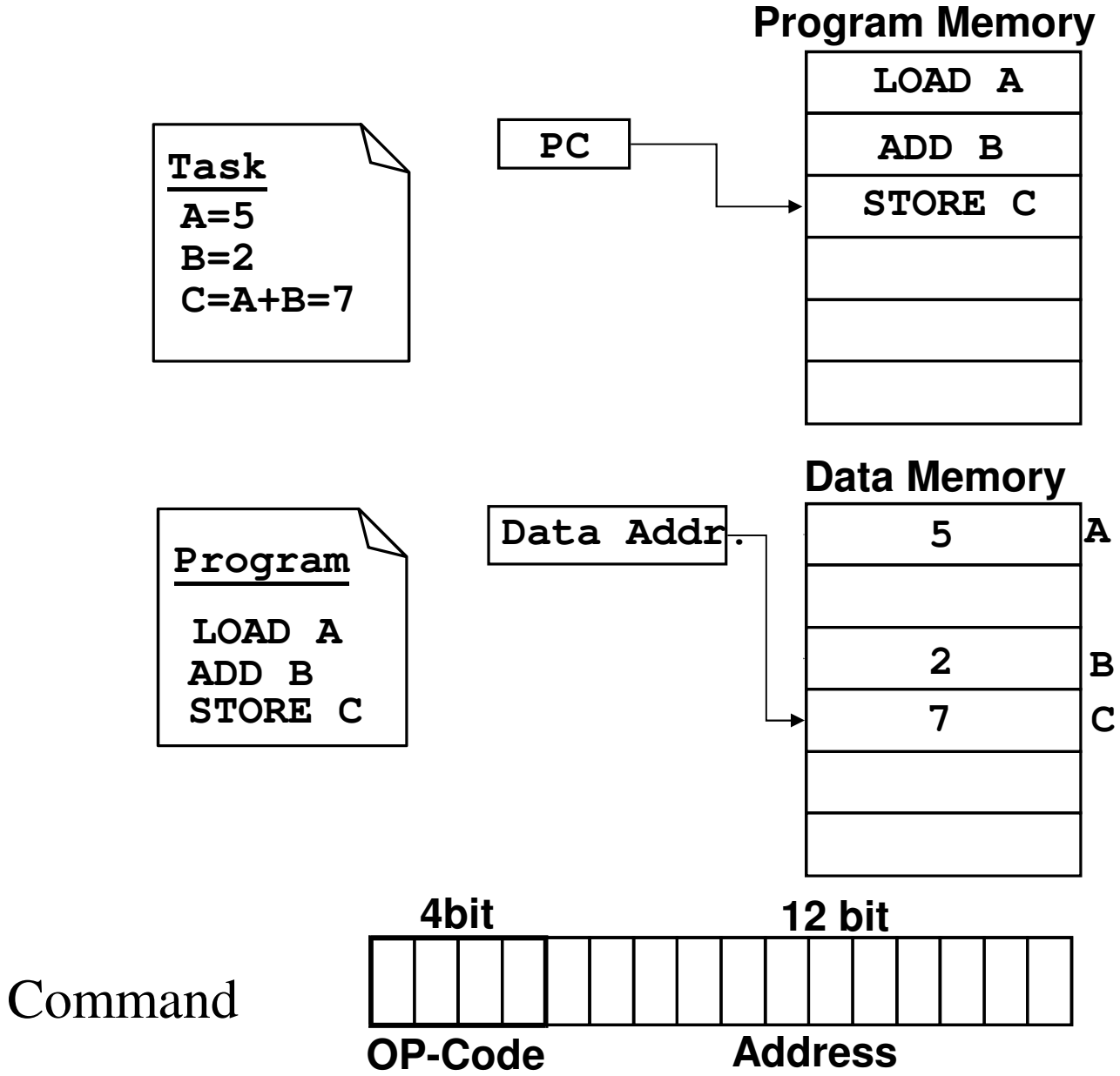
Program Counter (PC) - holds address of the next instruction to execute (a register)

Instruction Register (IR) - holds current instruction code being executed

Arithmetic Logic Unit (ALU) - performs operations on data

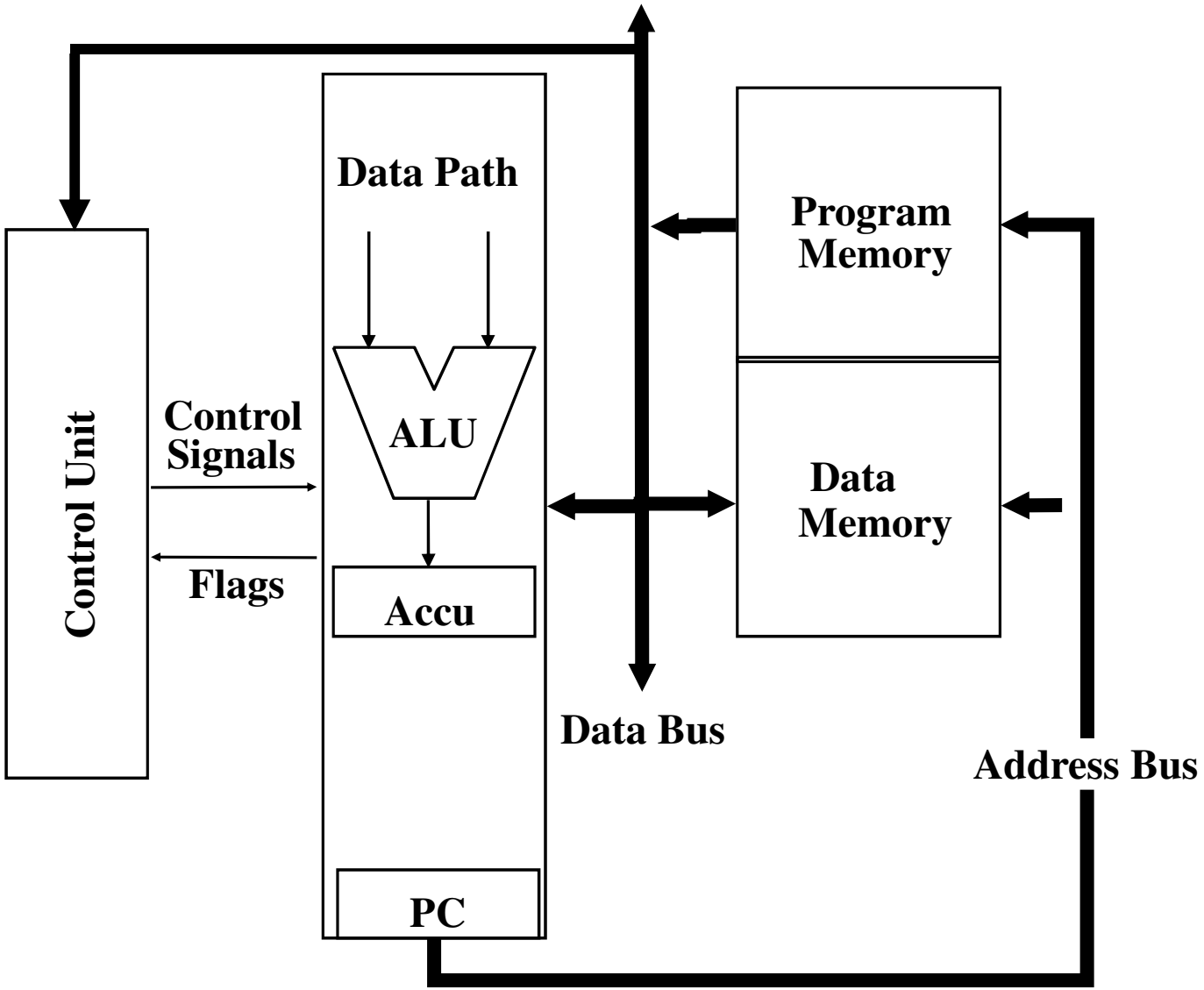
Accumulator (ACC) - holds data being processed (a register)

Program Flow

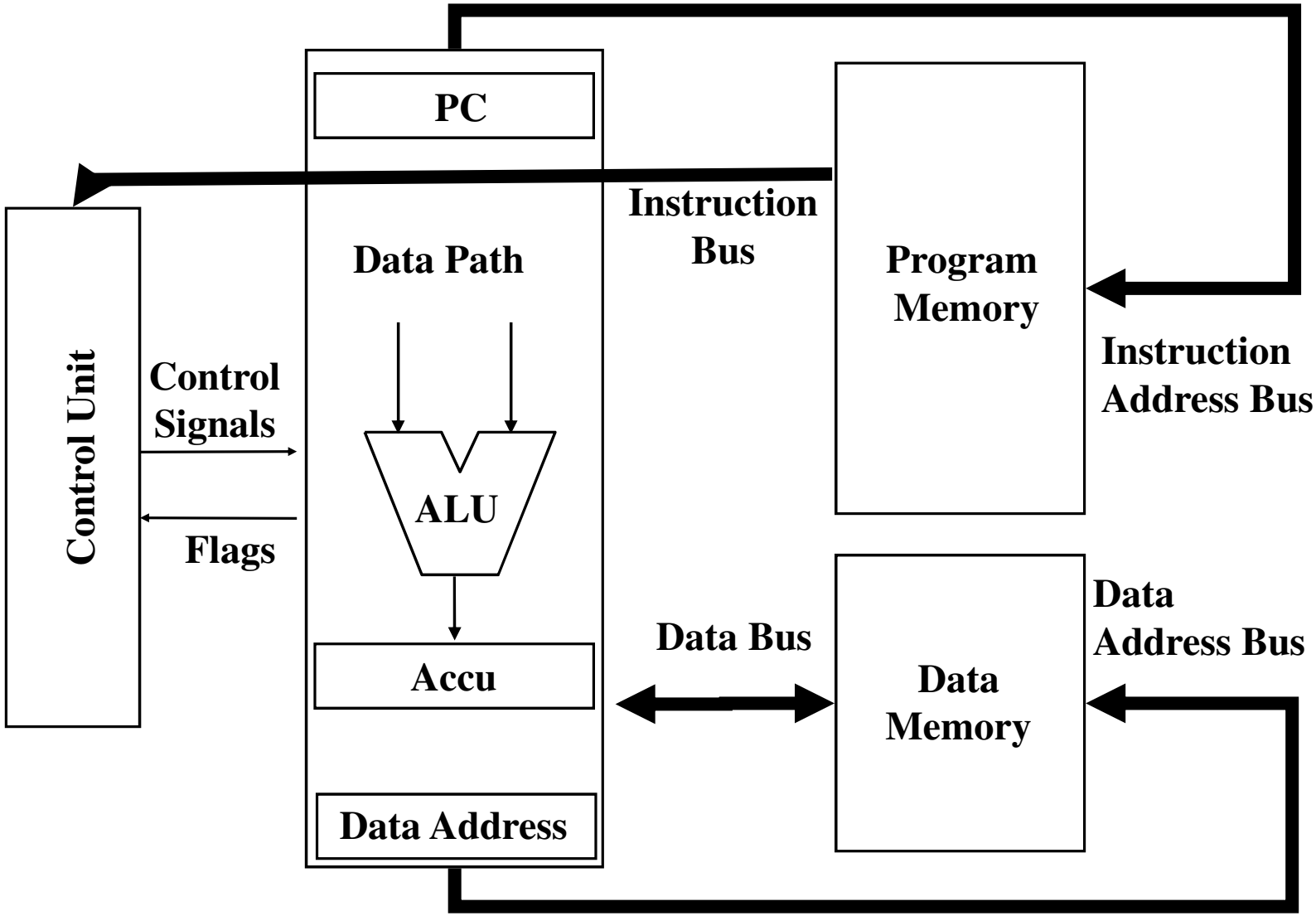


Command

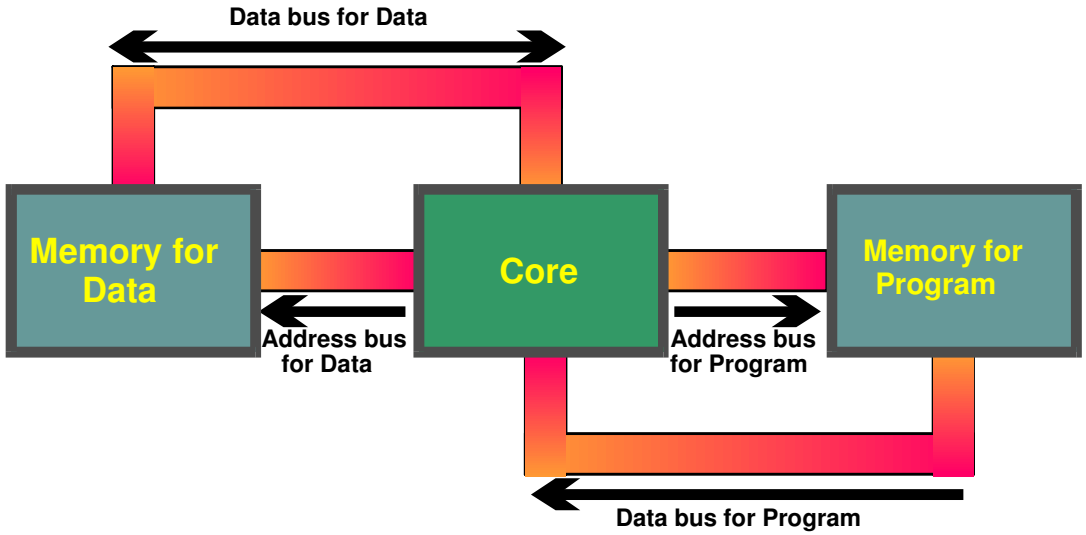
von Neumann Architecture



Harvard Architecture

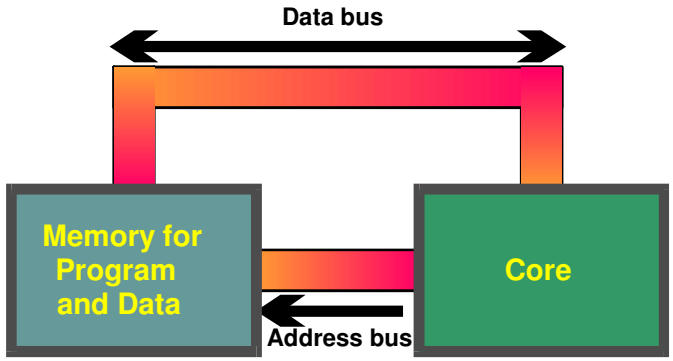


von-Neumann-/Harvard-Architecture



Harvard:

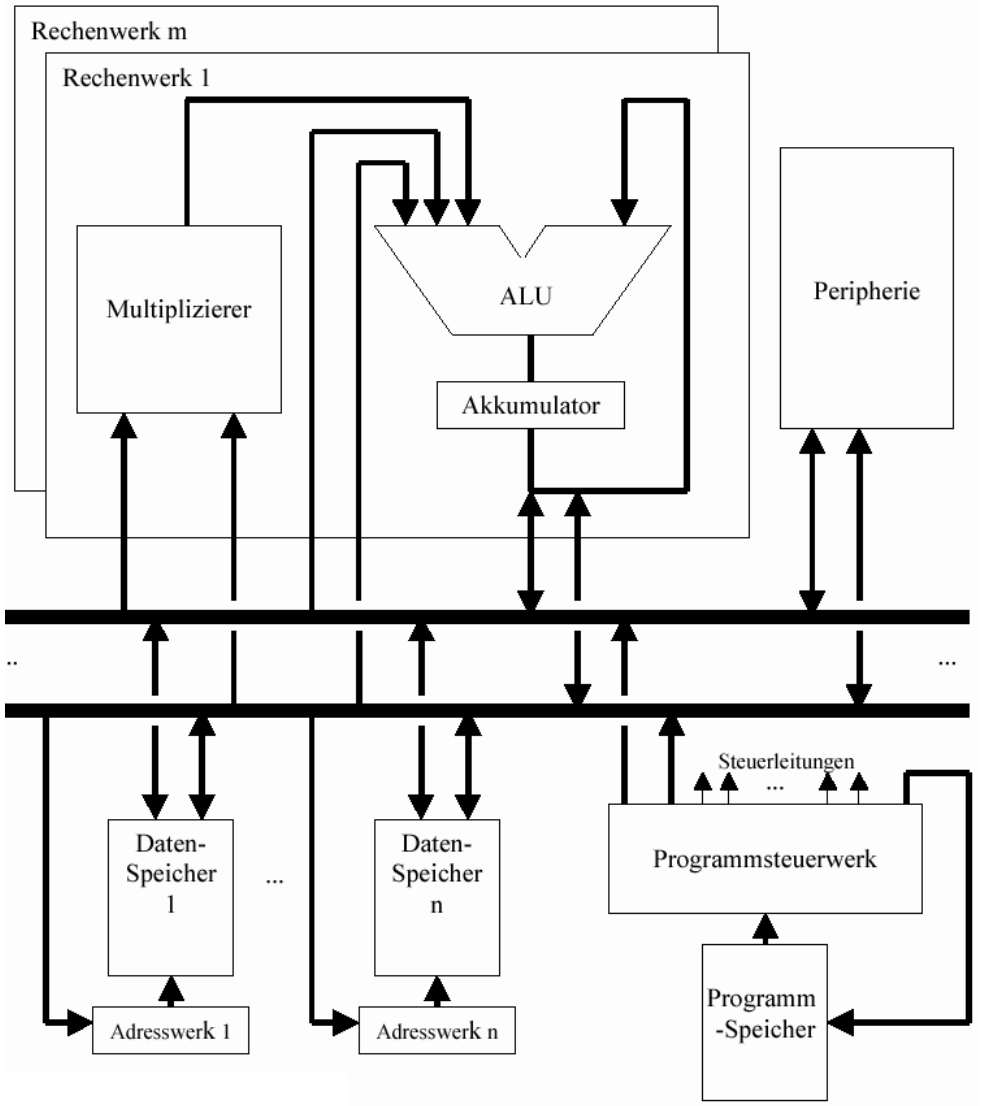
Seperated Busses for Data and Program



von-Neumann:

Common Busses for Data and Program

Digital Signal Processor

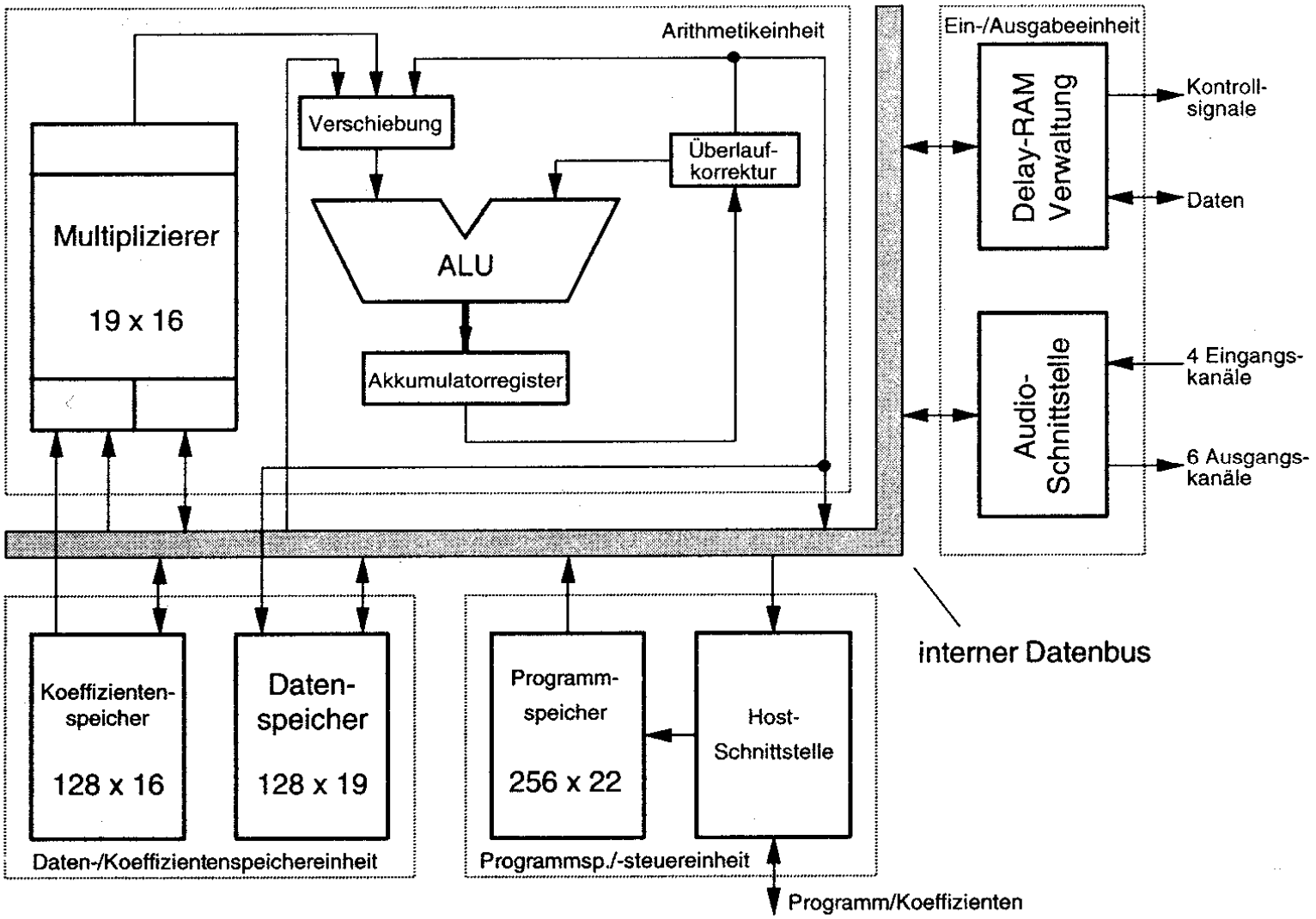


- Seperation Instructions and Data (Havard-Architecture), often several Data Busses
- high grade Pipelining
- High performance Arithmetic especially Multiply + Accumulate (MAC)
- highly user-controlled Parallelism
- Peripheral Signal Processing

Optimal for Fourier Transformation, Convolution, Digital Filter

Simple Signal Processor

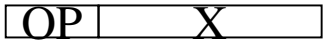
(NEC Audio-Signal Processor μPD6382)



Addressing / Operands

0 ⟶ Stack

1 ⟶ Accumulator

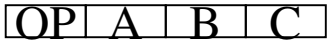


Accu=Accu+X

2 ⟶ Two Operands

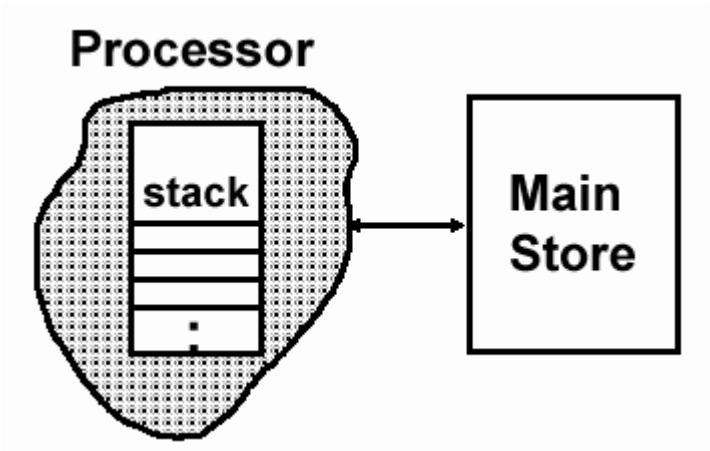
B=A+B

3 ⟶ Two Operands+ Target address



C=A+B

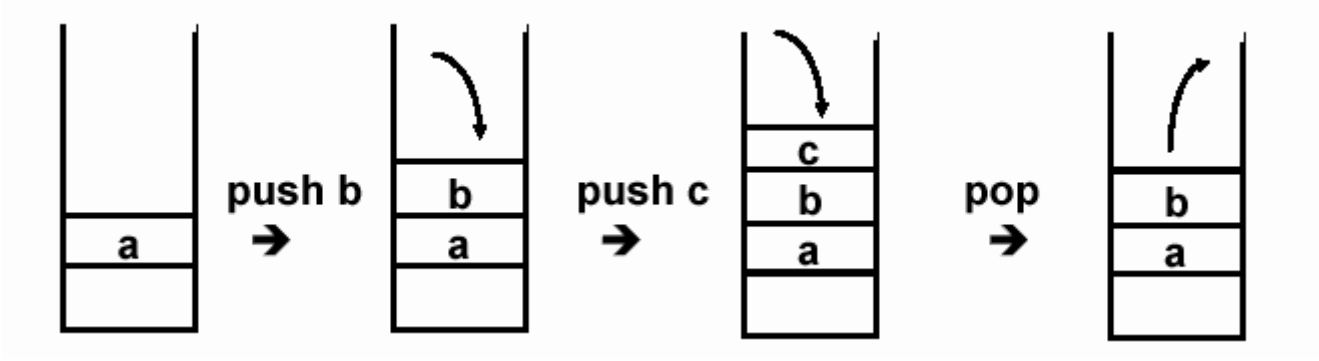
Stack Machine



Typical Operations:

Push

Pop



HW Organization: Register Memory in Processor

Operands Access

Memory	-	Memory	M[B] = M[B] + M[A]
Register	-	Memory	R1 = R1 + M[A]
Register	-	Register	R2 = R1 + R2

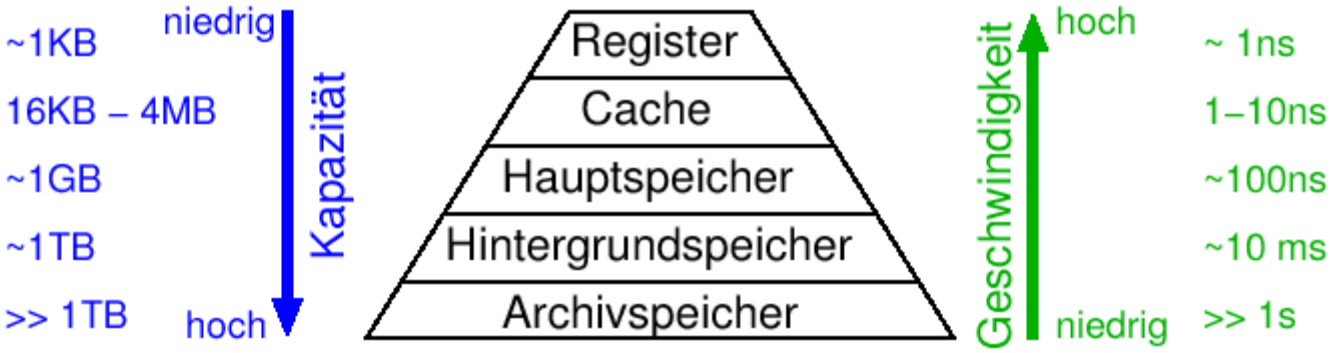
Register-Memory Machine

- Load R1, M[A]
- Add R1, M[B]
- Store R1, M[C]

Register-Register Machine

- Load R1, M[A]
- Load R2, M[B]
- Add R1, R2
- Store R2, M[C]

Memory Hierarchy



Implementation

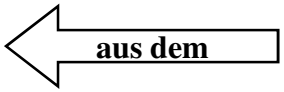
Register:		Flipflops
Cache:		SRAM
Main store	:	DRAM (SDRAM, DDR, Rambus)

Instruction Set

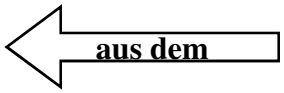
- Arithmetic / Logic
  (ADD, SUB, AND, OR, Shift L, R)
- Data Transfer
  (Load, Store $\Rightarrow$ Addressing Types)
- Control Flow
  (Jump, Subroutine, Branch)
  $\Rightarrow$ absolute, relative
  $\Rightarrow$ conditional : Z, N, C, ...

Instruction Sequence

- Program Counter 

Memory
- Command 

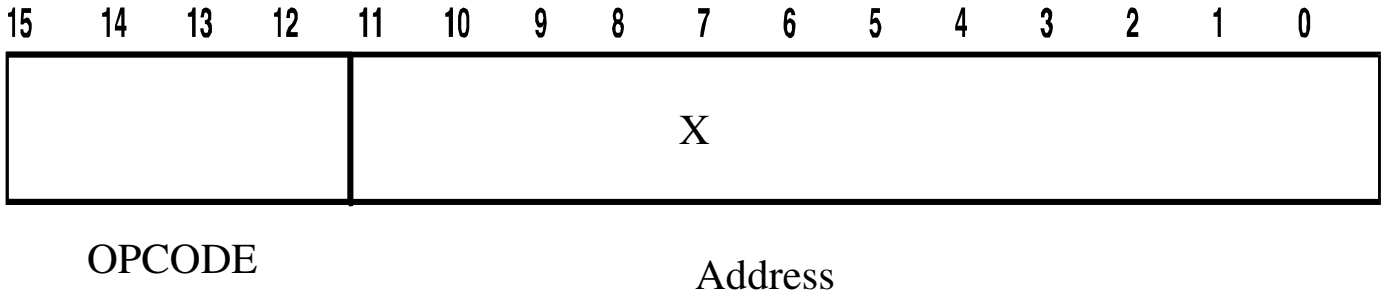
Memory
- Decoding of Command
- Operand Address 

Memory
- Operand 

Memory
- Execute Operation
- Save Results (Memory, Register)
- Increment Program Counter

UKM297 Instruction Set

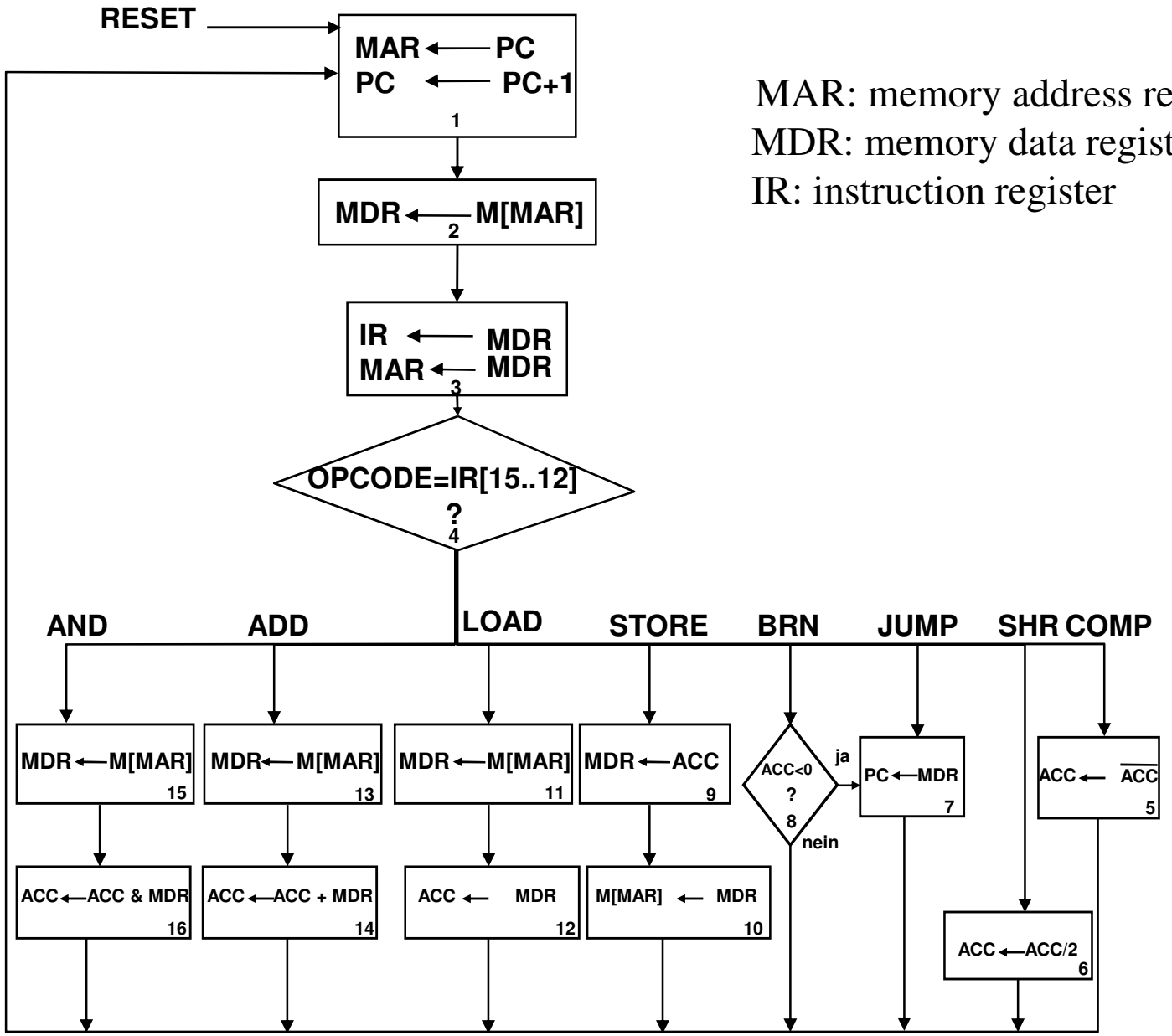
Command word of UKM 297



Commands of UKM297

COMP	ACC ← $\overline{ACC}$
SHR	ACC ← ACC/2 (Shift Right)
JUMP	PC ← x
BRN	PC ← x if ACC<0
LOAD	ACC ←M[x]
STORE	M[x] ← ACC
ADD	ACC ← ACC + MEM[x]
AND	ACC ← ACC & MEM[x]

Instruction Sequence



MAR: memory address register
MDR: memory data register
IR: instruction register