Computer and Operating Systems (COS)

Lecture 7

Overview of the contents

- Von Neumann architecture
- Simple computer model
- Control unit
- Instruction set design

Introduction to Computer Systems

Previously, calculating machines were designed and built to perform a **predetermined task**. All programming of such machines required manual reassembly of the wiring connections to the individual circuits, which was a cumbersome and often error-prone process.

Around 1950, John Von Neumann wrote some articles to introduce the **Von Neumann architecture**.

- "Theory and Organization of Complicated Automata" (1949)
- "The General and Logical Theory of Automata" (1951)
- "The Theory of Automata: Construction, Reproduction, Homogeneity" (1952).

They formed the basis of the structure of the modern computers.

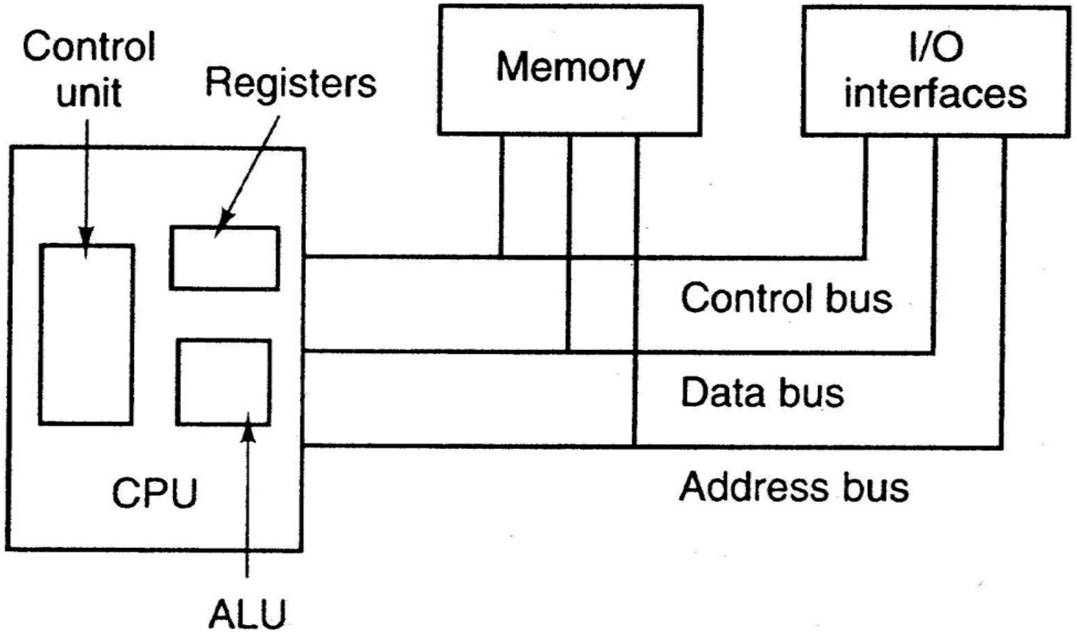
The basic idea of the Von Neumann architecture lies in the ability to store program instructions in memory along with the data on which these instructions operate.

Here, program and data are in the same memory. It is the program that decides what should happen. The computer with Von Neumann architecture **does not have a predetermined purpose in its design**.

Introduction to Computer Systems

Von Neumann architecture: Basic idea

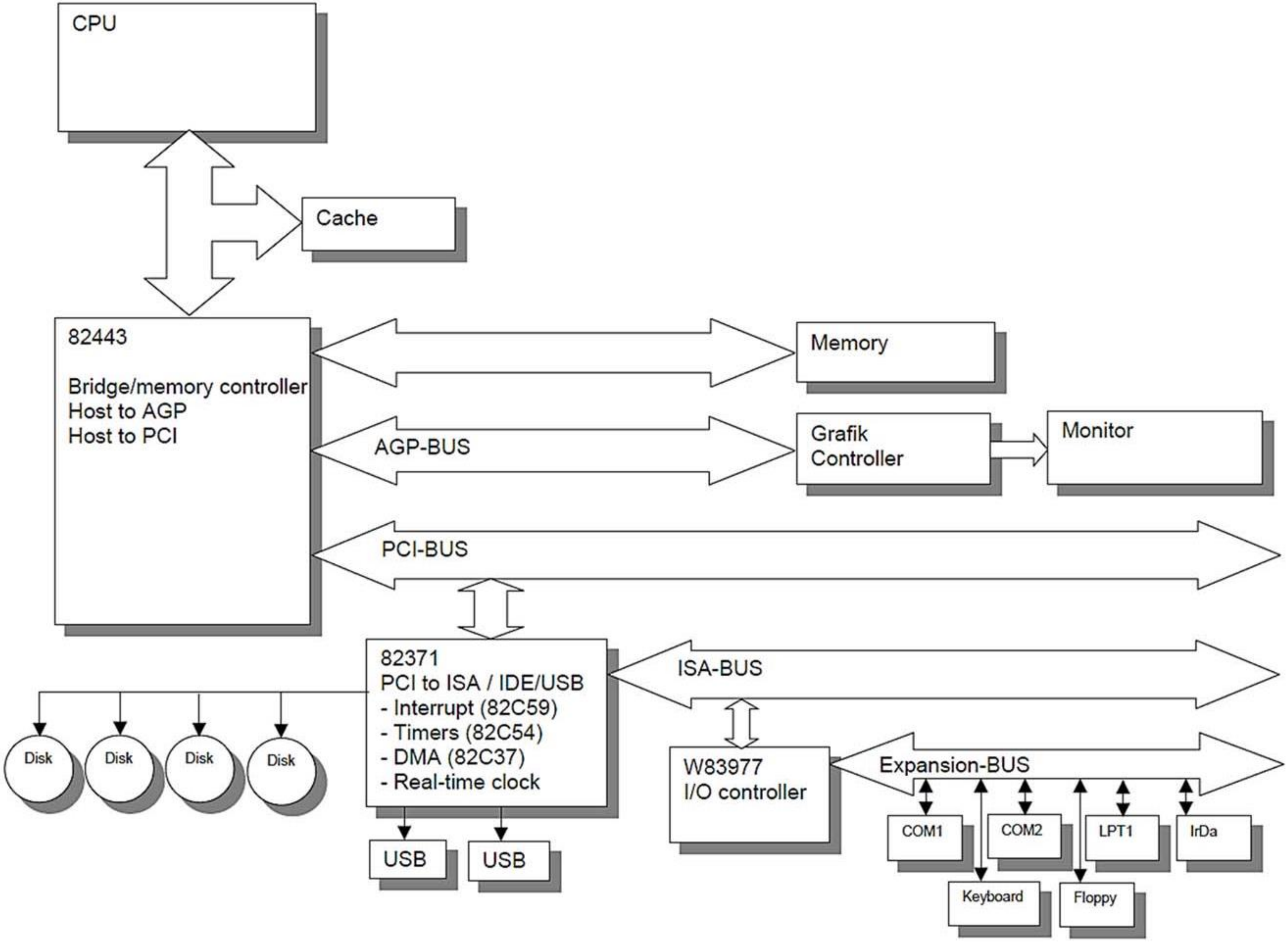
The Von Neumann architecture is composed of three independent components (or subsystems): a **Central Processing Unit (CPU)**, **Memory** and an **Input / Output device (I / O interface).**



1. The CPU is the “brain” of the computer system and contains three main components: **Control unit**, one or more **Arithmetic Logic Units (ALUs)** and a varying number of **Registers**.
2. The memory is used to store program instructions and data. Two commonly used types of memory are **Random Access Memory (RAM)** and **Read-Only Memory (ROM)**.
3. The I / O interface enables the computer's memory to exchange information with the outside world.

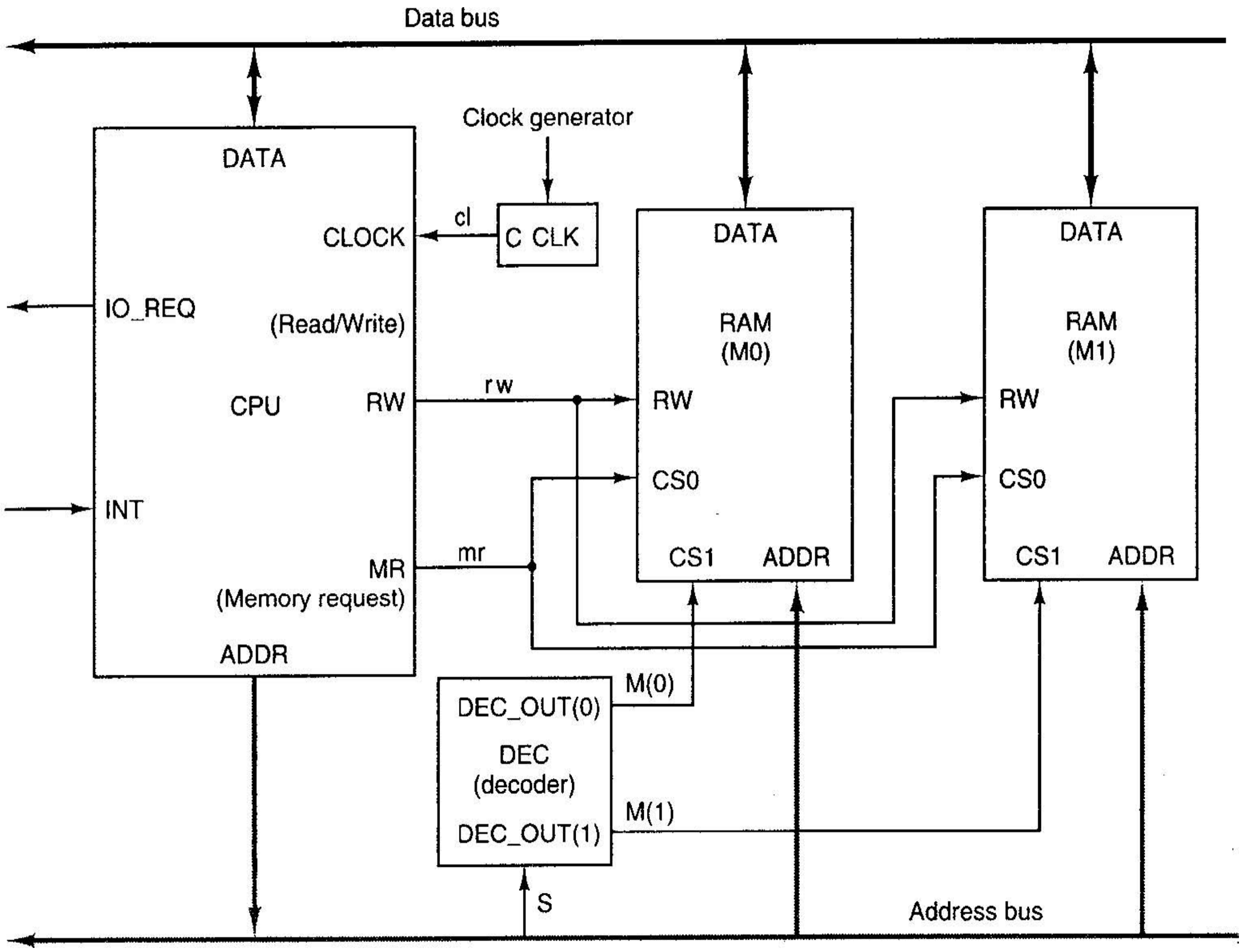
Introduction to Computer Systems

Von Neumann architecture: Example of a Pentium II Motherboard



Introduction to Computer Systems

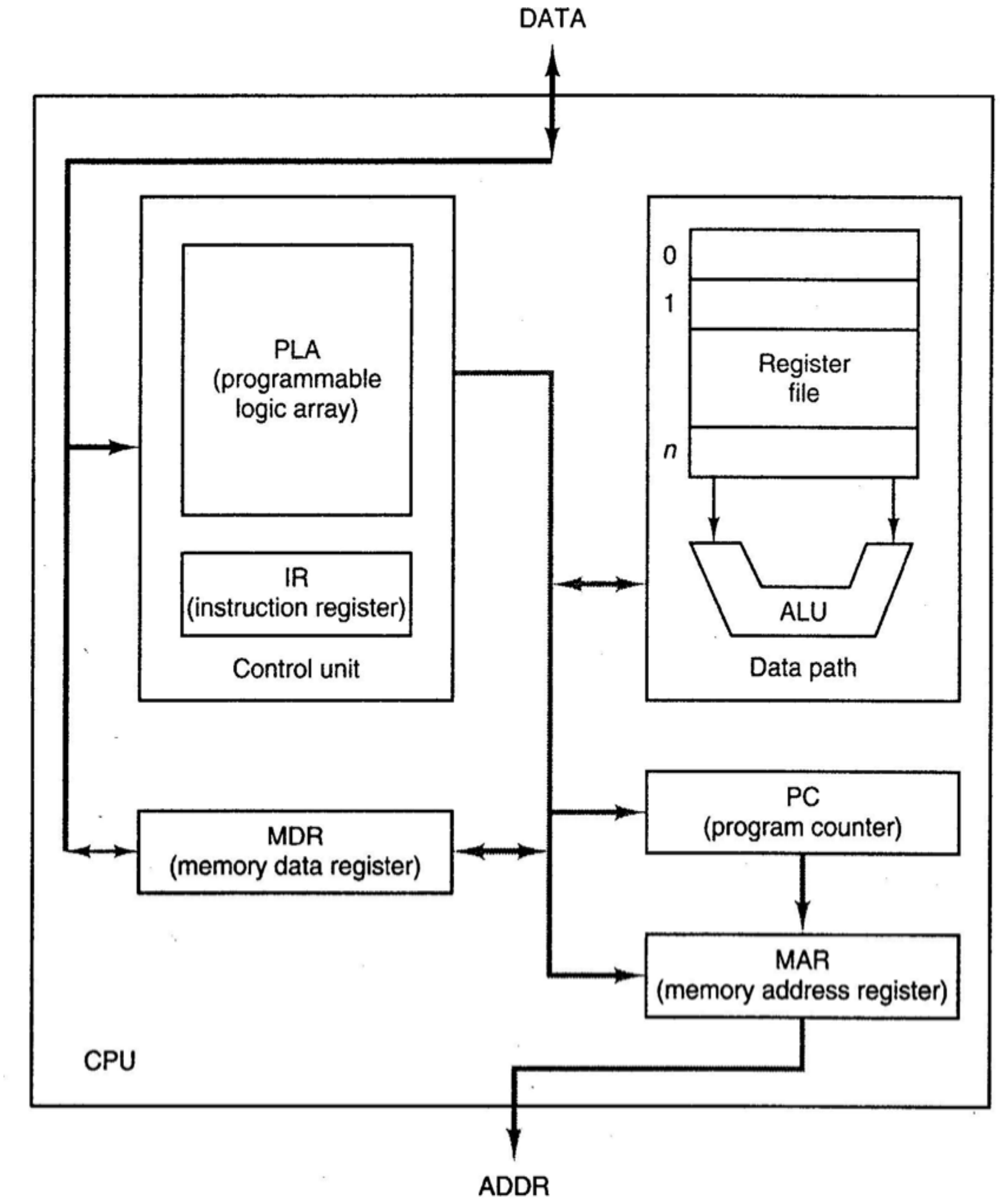
Simple computer model



This microcomputer contains a CPU, a clock generator, a decoder and two memory modules. Each memory module contain 8 addresses, each of them is of 8-bit long.

Introduction to Computer Systems

Simple computer model: CPU



Both data and instructions are retrieved via the data bus from the same memory.

Data is retrieved to **MDR**
Instructions to be decoded and executed are downloaded to **IR**

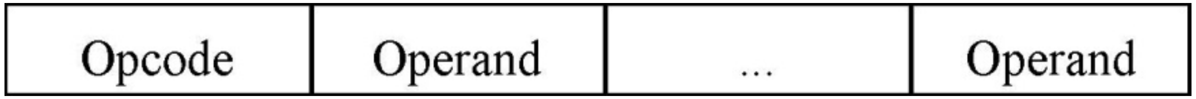
MAR designates the address in memory to be read or written.

PC keeps track of program flow by storing the address of the next instruction to be executed.

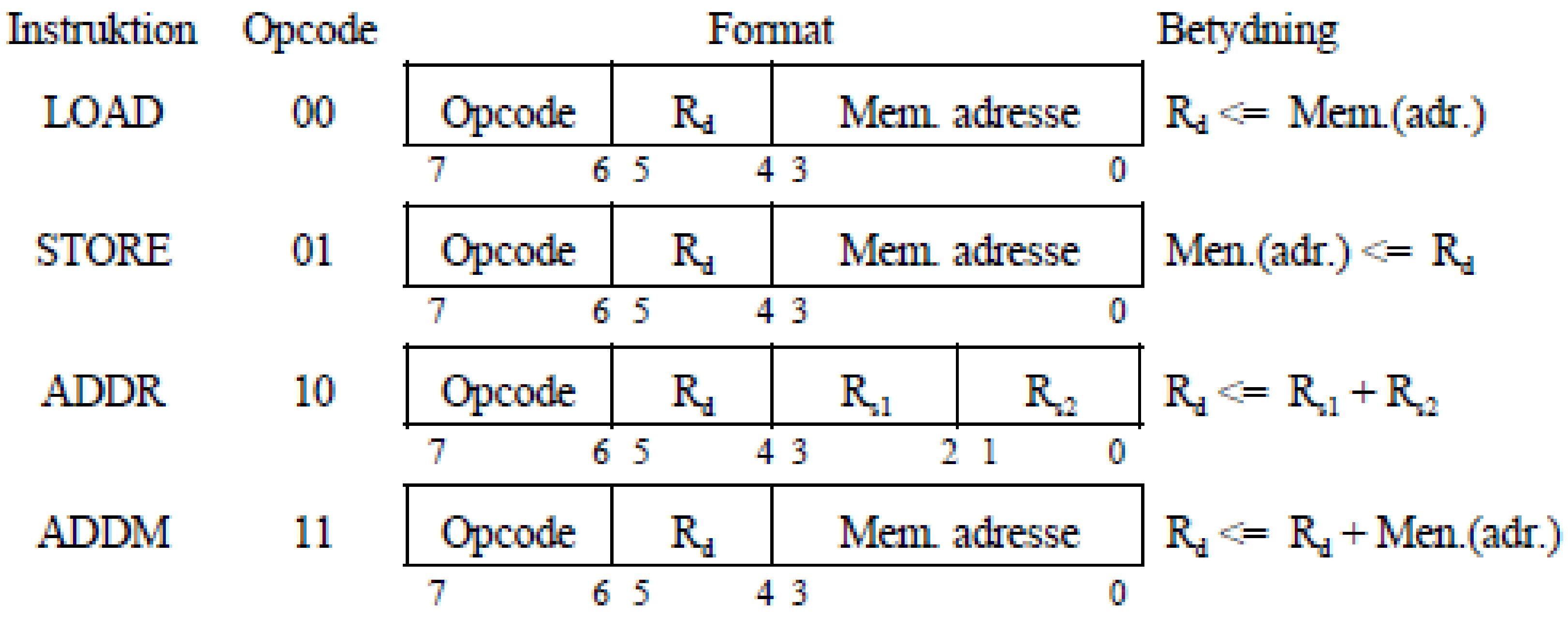
Introduction to Computer Systems

Simple computer model: Instructions

The general format of an instruction



Assume that our microprocessor has only four different instructions

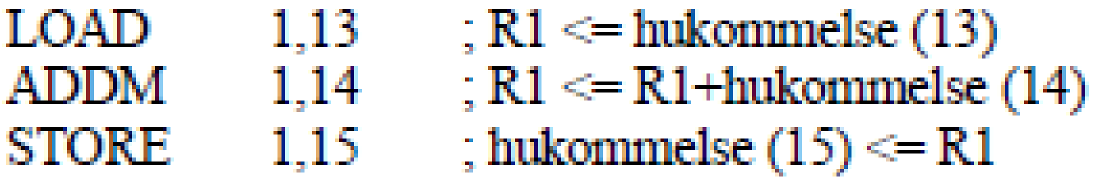


Instruktion	Opcode	Format				Betydning
LOAD	00	Opcode	R_d	Mem. adresse		R_d <= Mem.(adr.)
		7 6	5 4	3 0		
STORE	01	Opcode	R_d	Mem. adresse		Men.(adr.) <= R_d
		7 6	5 4	3 0		
ADDR	10	Opcode	R_d	R_{s1}	R_{s2}	R_d <= $R_{s1} + R_{s2}$
		7 6	5 4	3 2	1 0	
ADDM	11	Opcode	R_d	Mem. adresse		R_d <= R_d + Men.(adr.)
		7 6	5 4	3 0		

Introduction to Computer Systems

Simple computer model: a program example

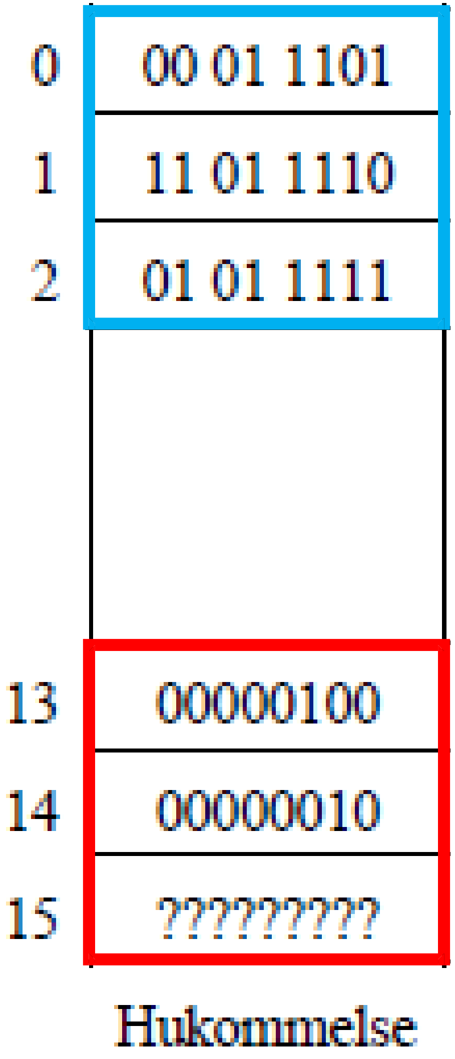
Illustrate the execution of a simple program in the microcomputer. Assume that the program is supposed to add two numbers from addresses 13 and 14 together and saves the result at address 15. If we use the available instructions, the program can look like this:



```
LOAD     1,13     ; R1 <= hukommelse (13)
ADDM     1,14     ; R1 <= R1+hukommelse (14)
STORE    1,15     ; hukommelse (15) <= R1
```

Introduction to Computer Systems

Simple computer model: Program and data in memory together



0	00 01 1101
1	11 01 1110
2	01 01 1111
13	00000100
14	00000010
15	?????????

Hukommelse

Program

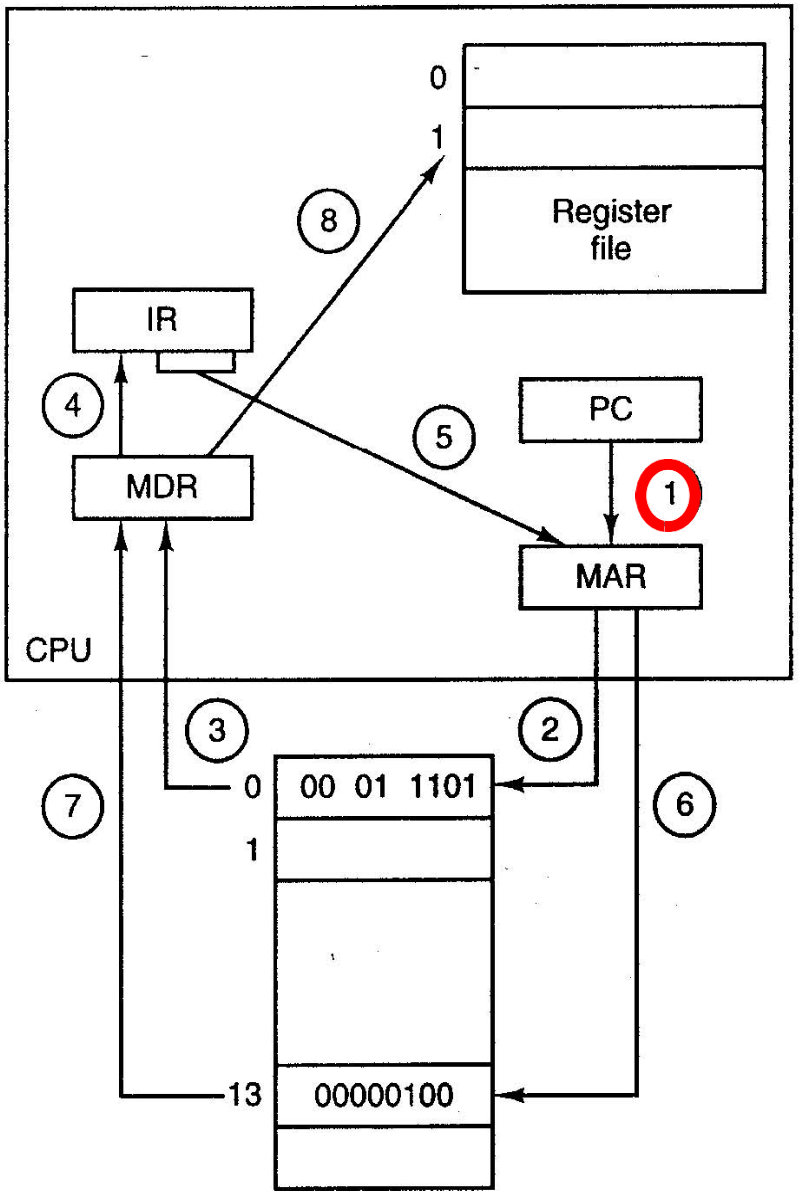
```
LOAD     1,13     ; R1 <= hukommelse (13)
ADDM     1,14     ; R1 <= R1+hukommelse (14)
STORE    1,15     ; hukommelse (15) <= R1
```

Data

The first three addresses are used to store the instructions while the last three addresses are used to store data. The program and data are stored in the memory together.

Introduction to Computer Systems

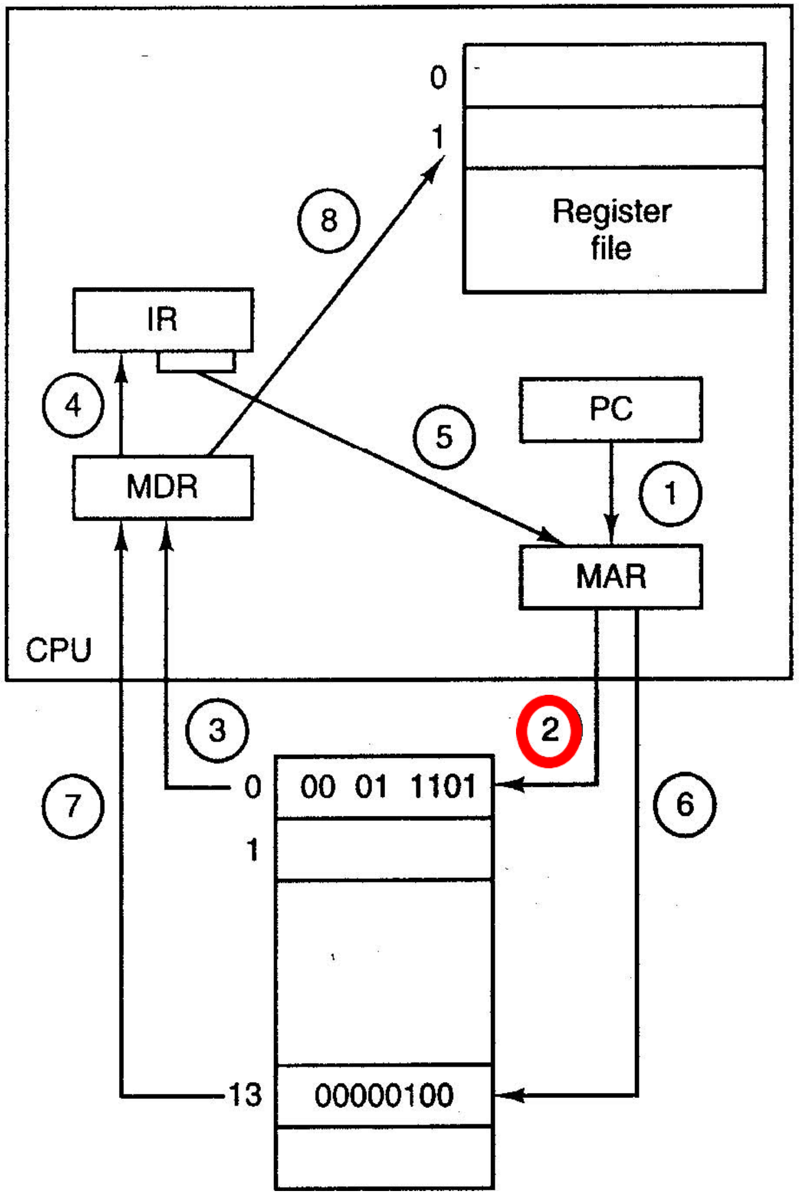
Simple computer model: Program execution of the LOAD instruction



The content in Program Counter (**PC**=0) is copied to the Memory Address Register (**MAR**)

Introduction to Computer Systems

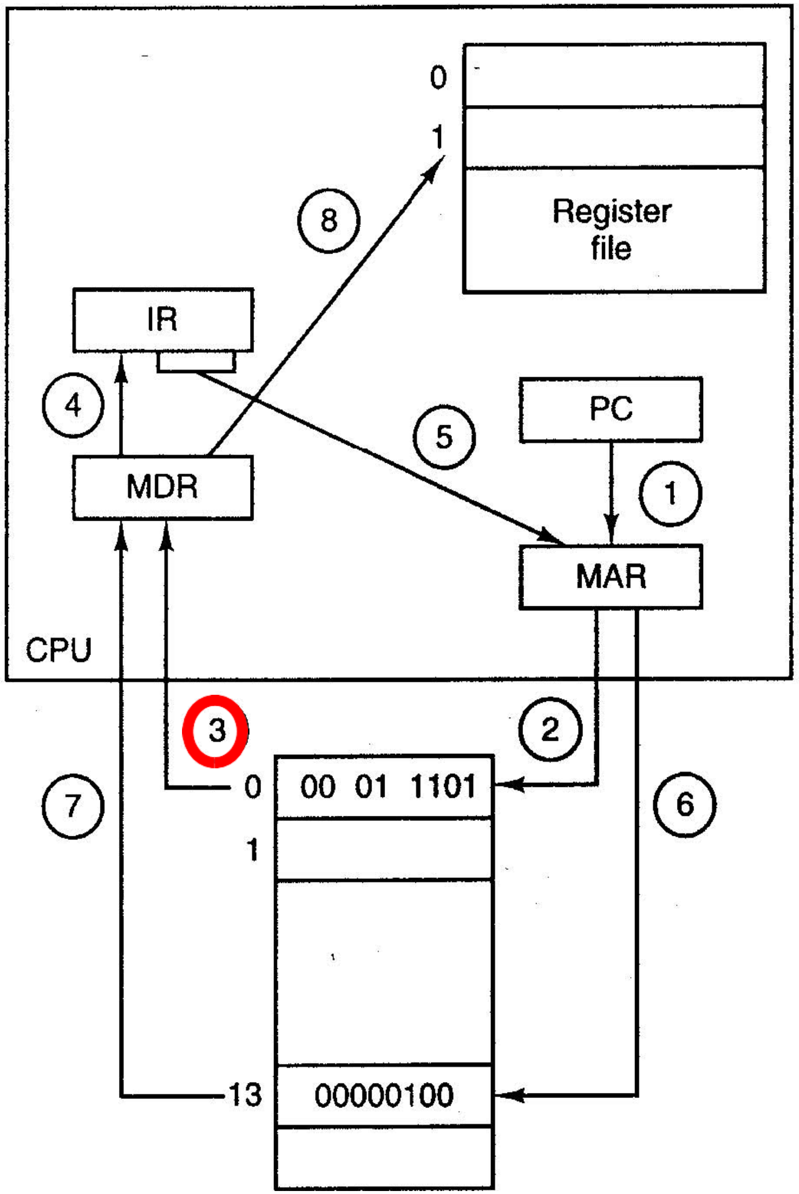
Simple computer model: program execution of the LOAD instruction



Memory Address Register (**MAR**) points to the first instruction in the memory (the first cell), i.e., LOAD instruction

Introduction to Computer Systems

Simple computer model: program execution of the LOAD instruction

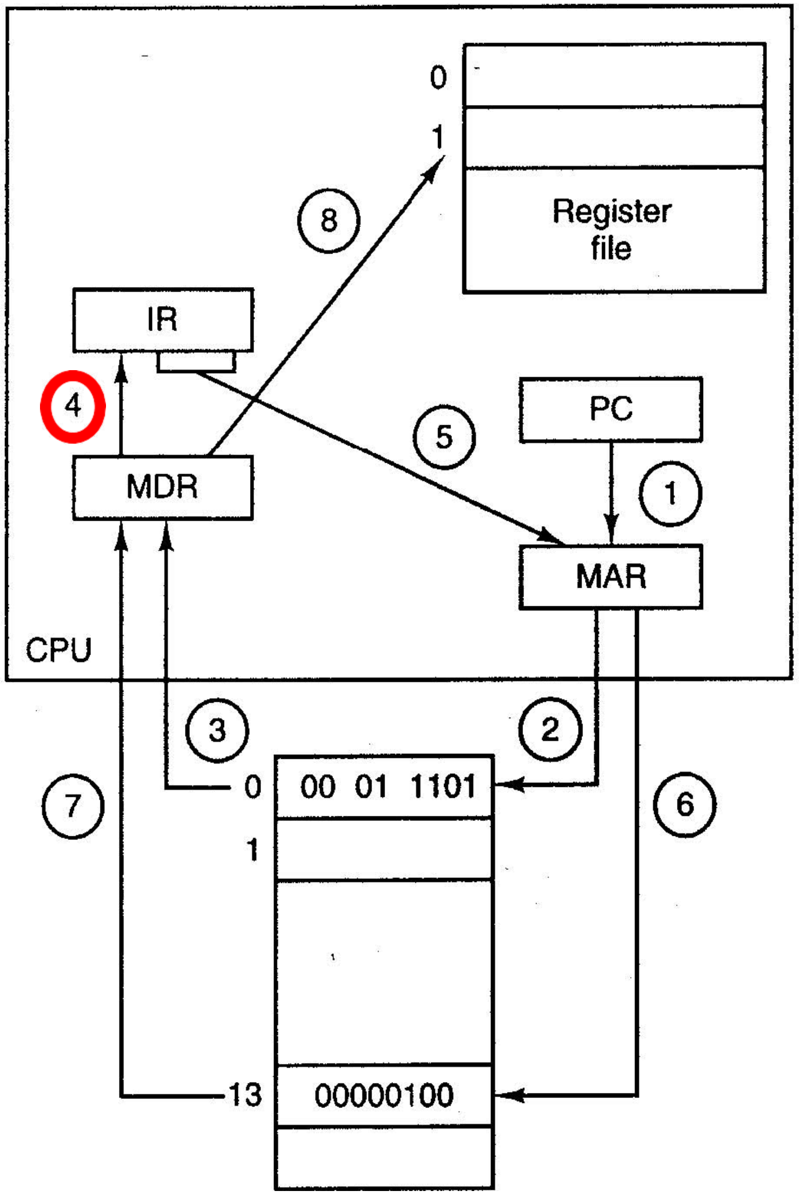


The control unit requests reading the corresponding memory cell and brings the content of address 0 into the Memory Data Register (**MDR**)

At the same time, the value in **PC** is increased by 1.

Introduction to Computer Systems

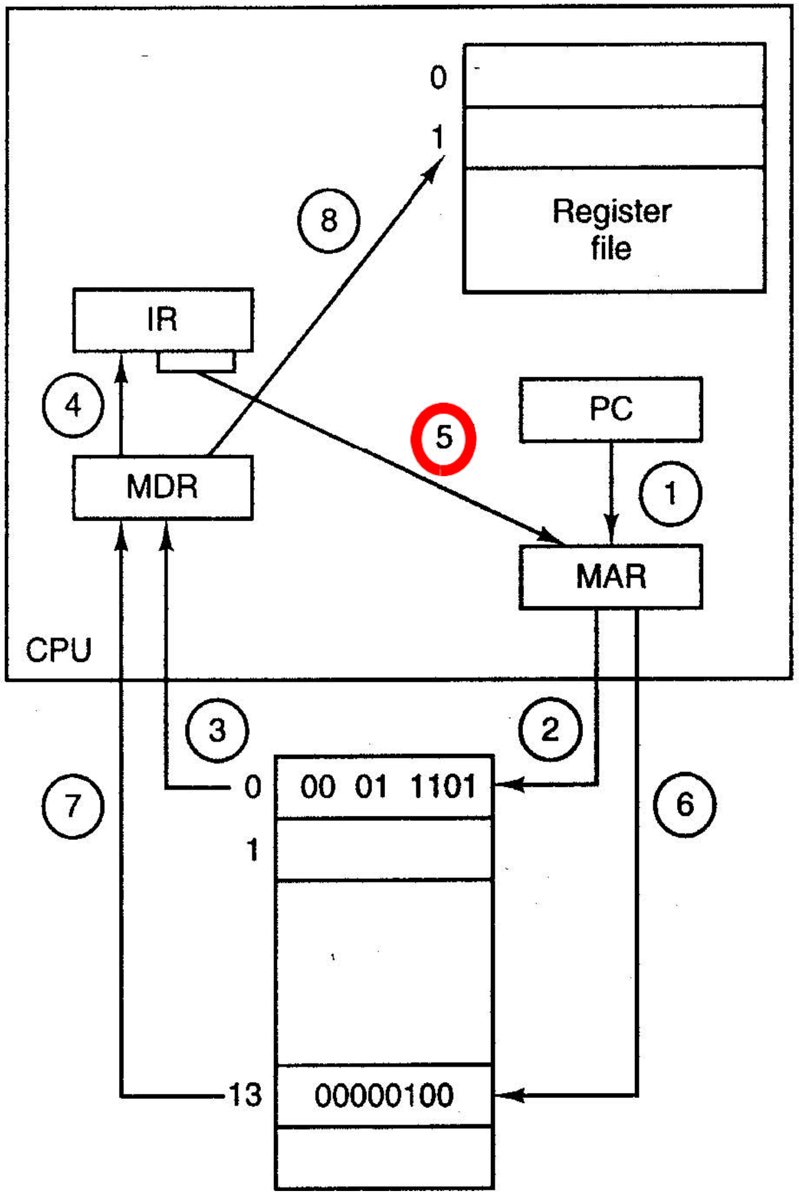
Simple computer model: program execution of the LOAD instruction



The content of **MDR** is copied to the Instruction Register (**IR**) where the instruction to be executed is decoded, i.e., 00 = the LOAD instruction.

Introduction to Computer Systems

Simple computer model: program execution of the LOAD instruction

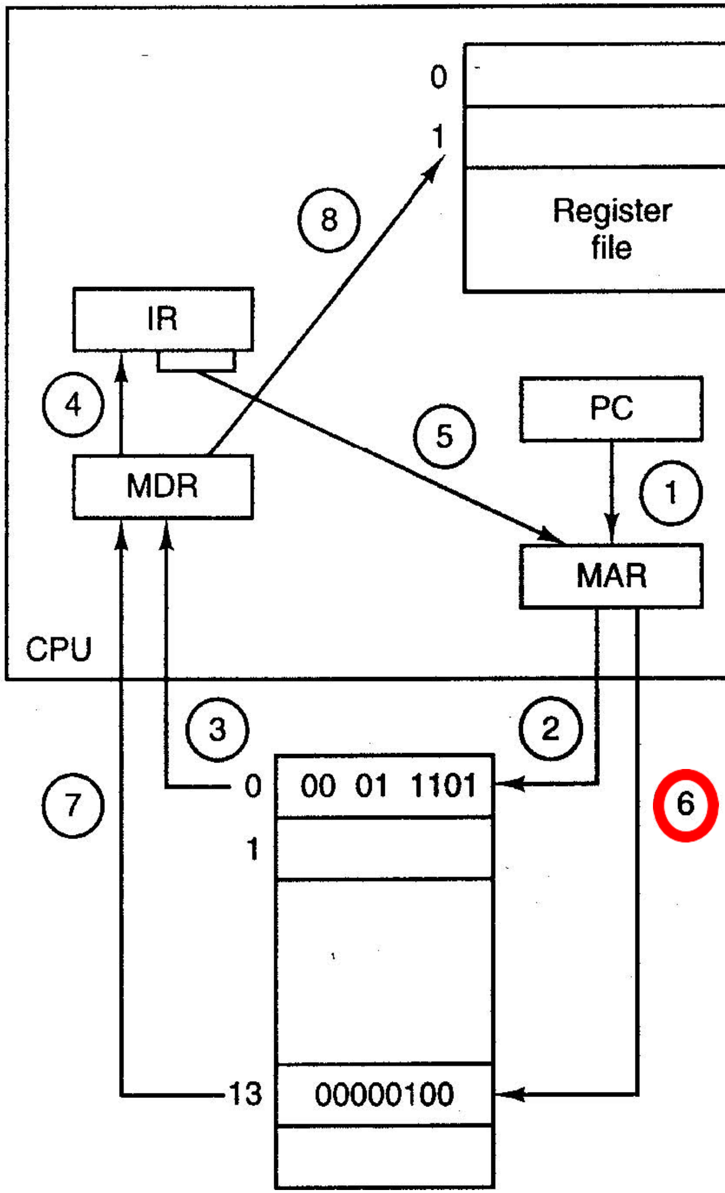


The two *MSB* (00) indicate that this is the *LOAD* instruction

Therefore, the control unit will now copy the 4 *LSBs* from the **IR** to the **MAR** (1101 or 13)

Introduction to Computer Systems

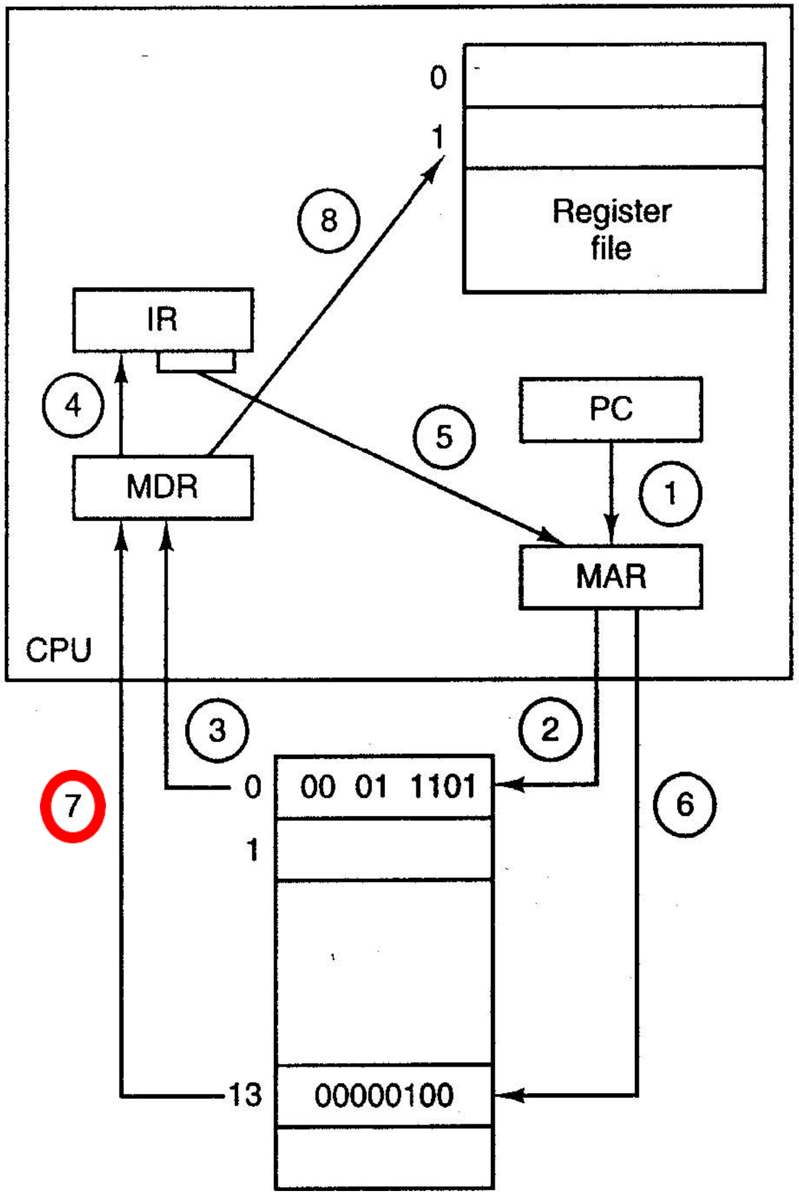
Simple computer model: program execution of the LOAD instruction



MAR now points to address 1101 or 13

Introduction to Computer Systems

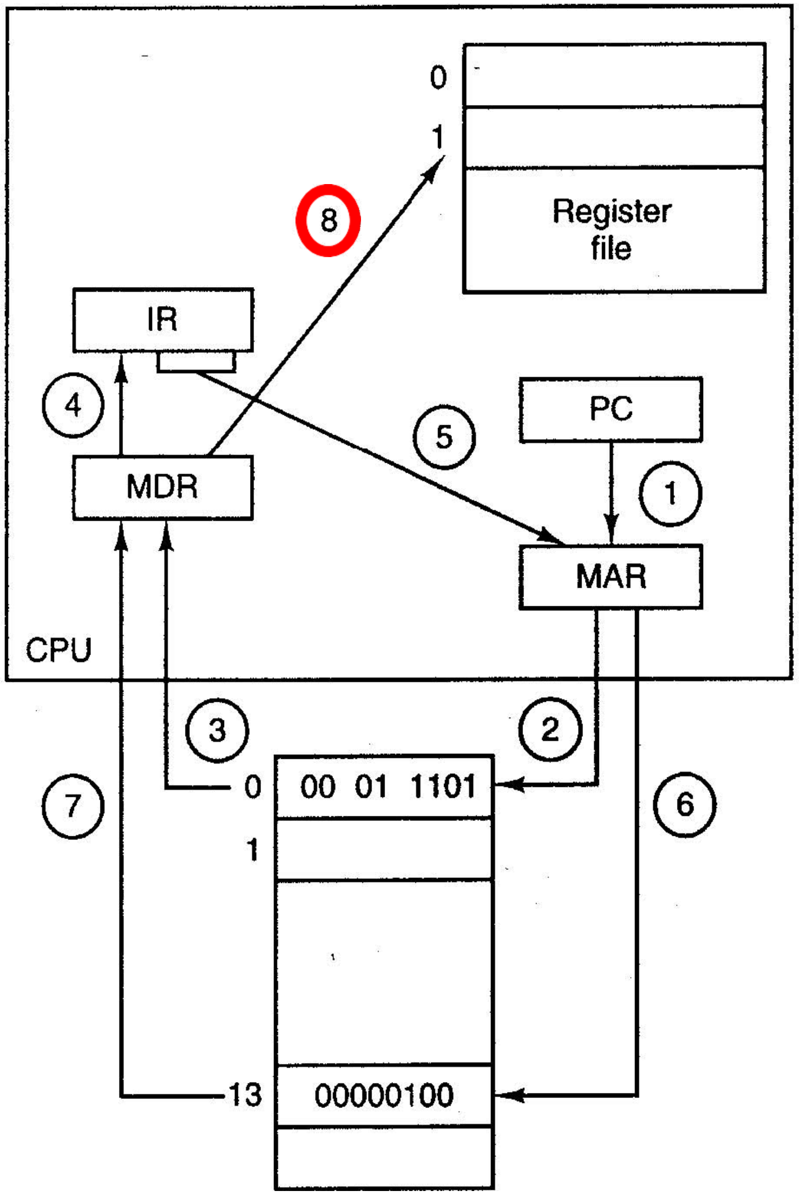
Simple computer model: program execution of the LOAD instruction



The control unit requests reading the corresponding memory cell and brings the contents of address 13 into the **MDR**

Introduction to Computer Systems

Simple computer model: program execution of the LOAD instruction

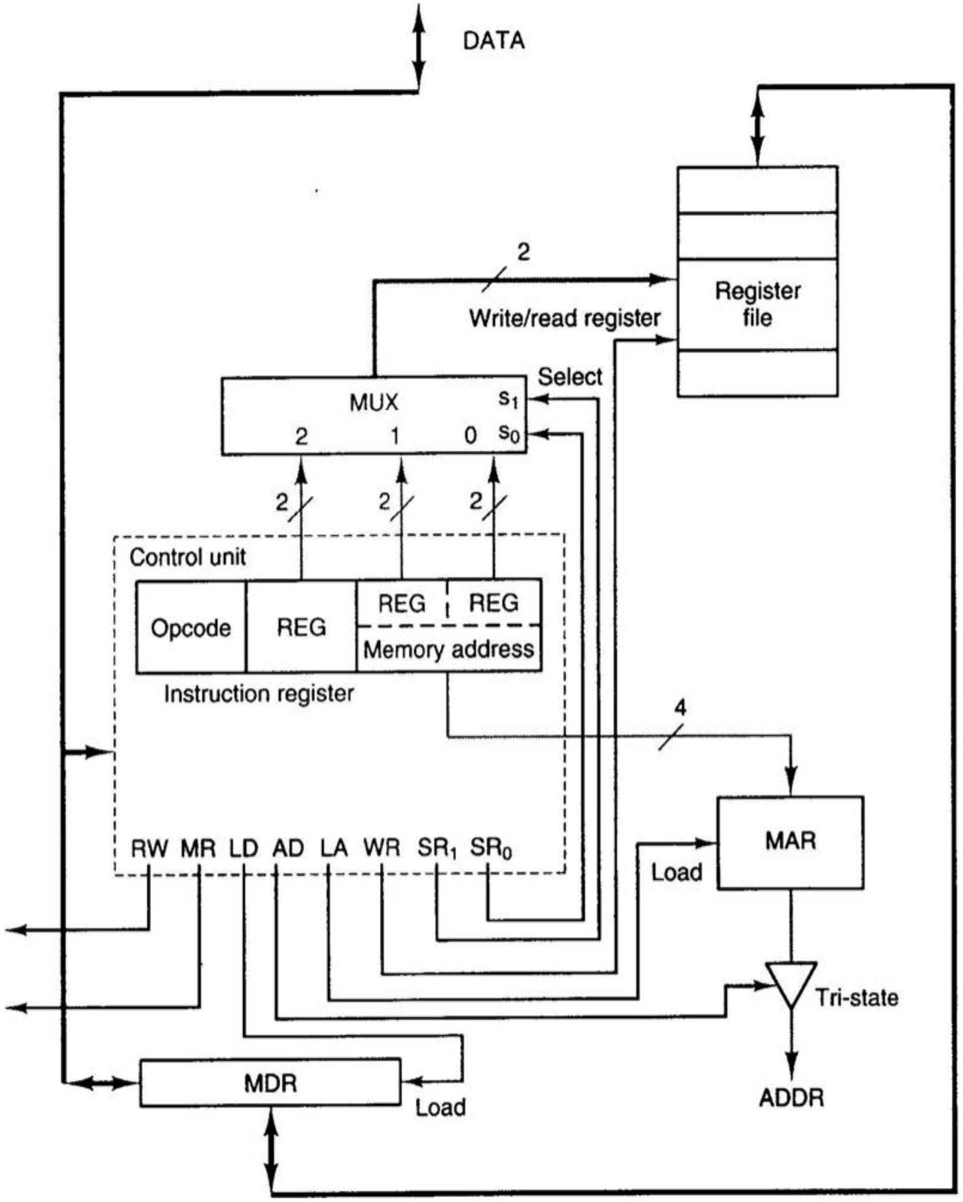


Based on the value of the **IR** register in bits 4 and 5, the **MAR** register is copied to address 1 in the register file block.

(IR = 00 **01** 1101)

Introduction to Computer Systems

Simple computer model: the control unit



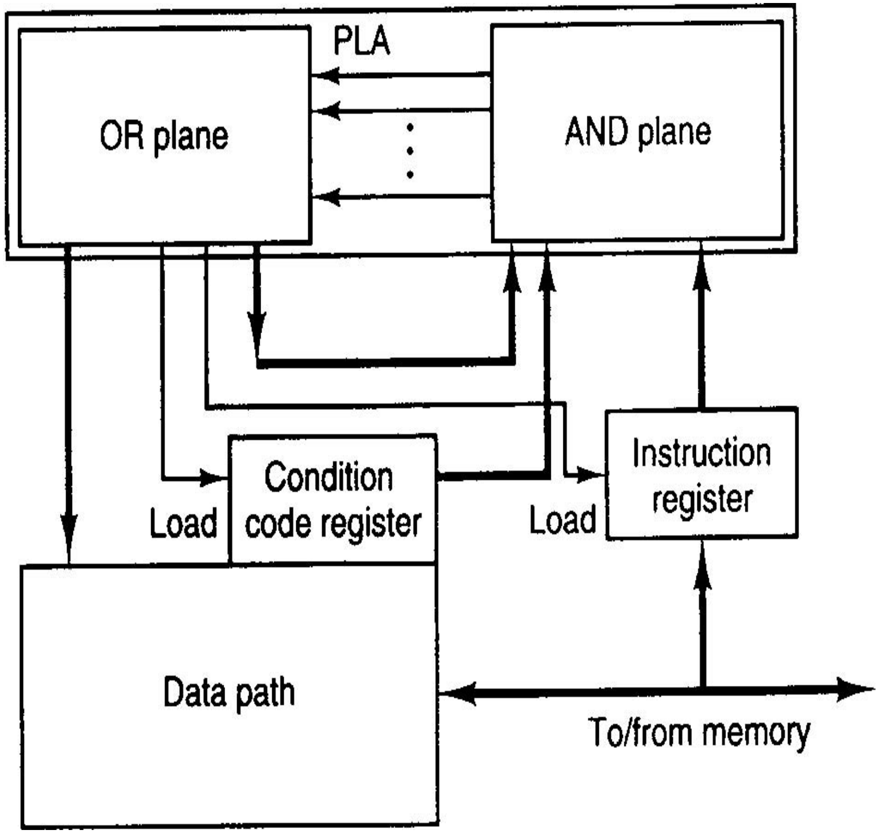
From the control unit, there are connections that control the various registers of the CPU.

It is the control unit that executes a sequence of operations that will correspond to a specific instruction. E.g., the LOAD instruction

This sequence of operations, which implement the instruction, are called: **micro-instructions**

Introduction to Computer Systems

Simple computer model: The control unit of **Hardwired Circuit** design



Here, a **Programmable Logic Array (PLA)** is used to implement a hardwired circuit for the control unit.

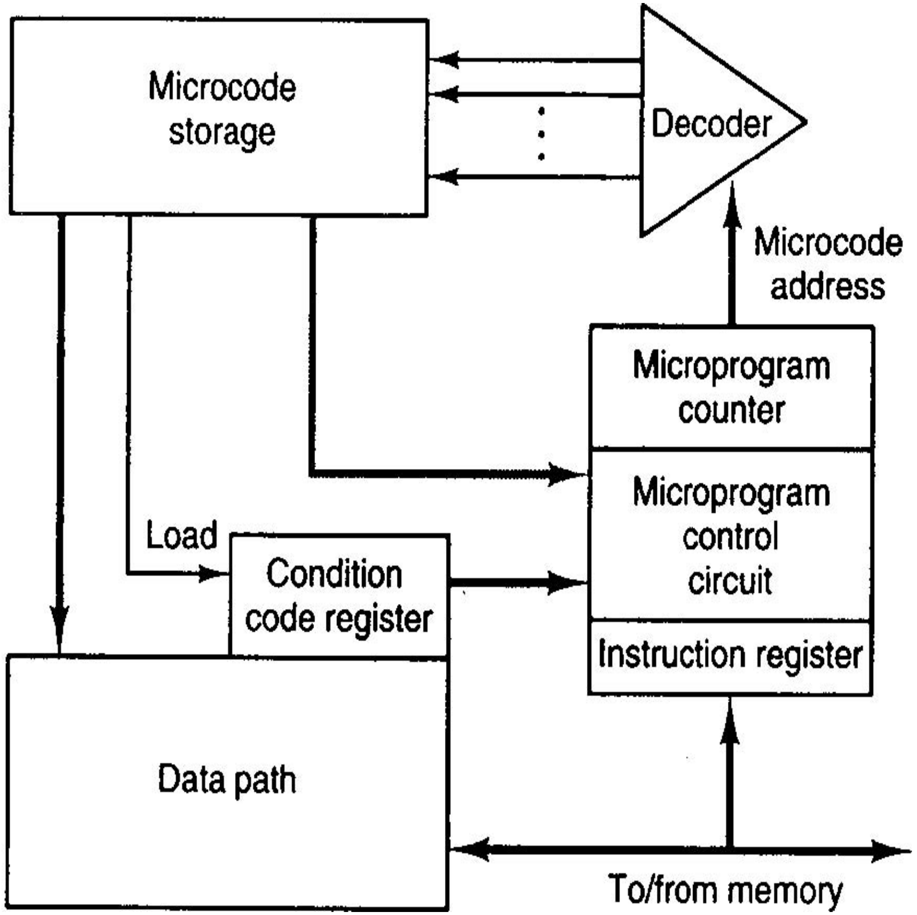
Disadvantage: Inflexible

later changes in the control unit require a total redesign of the entire circuit.

Advantage: Efficient

Introduction to Computer Systems

Simple computer model: the control unit of **Microprogram** design



Here, each instruction is executed as a form of subroutine consisting of microinstructions

Advantage: Flexible

A computer in a computer

Disadvantage: Less efficient

Introduction to Computer Systems

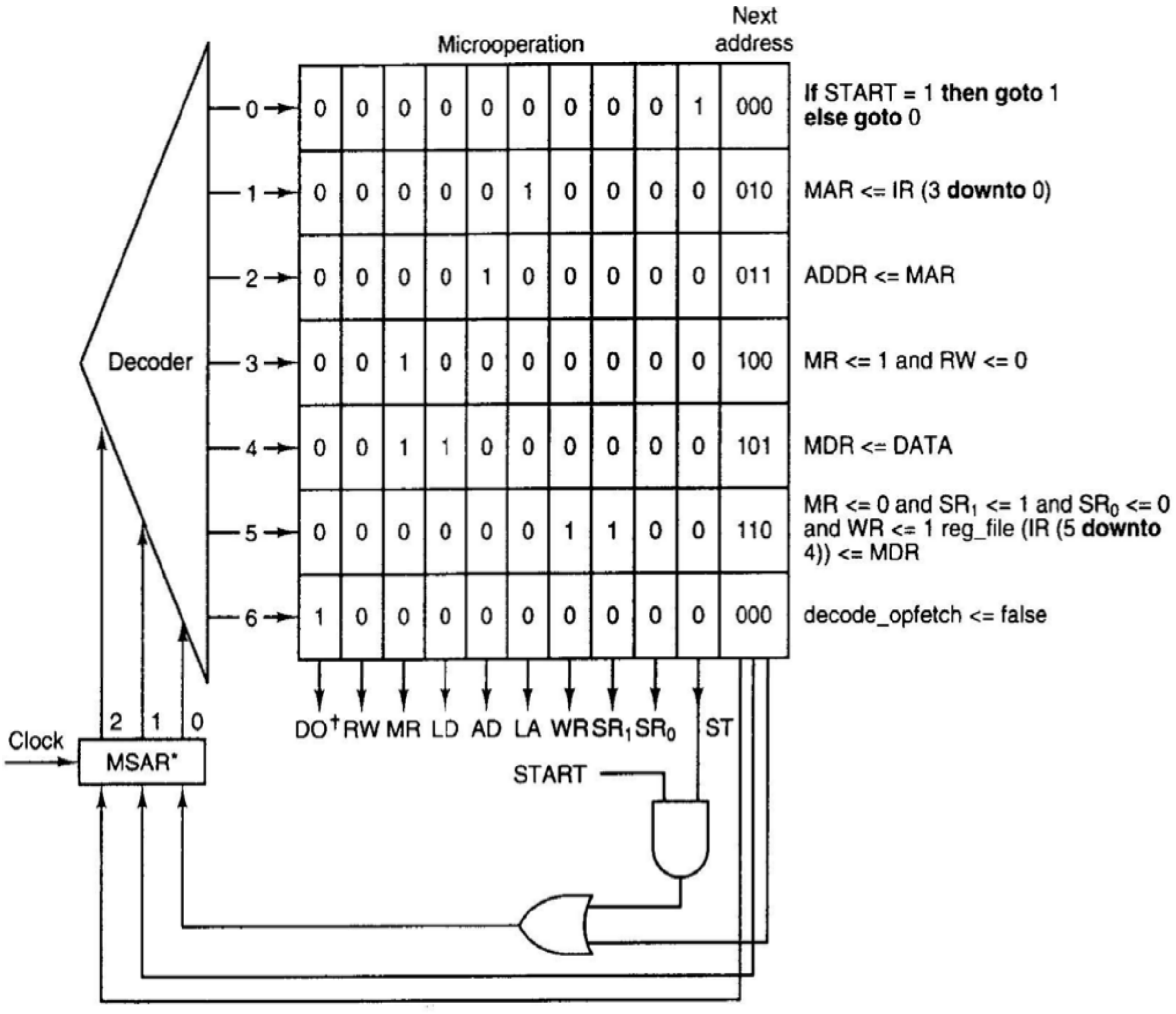
Simple computer model: Design of the control unit

The following considerations should be made when designing microinstruction words:

- **Minimizing** the size of the microcode memory in width, i.e., microword size.
- **Minimizing** the size of the microprogram, i.e., the size of the microcode memory in length.
- **Maximizing** the flexibility of adding and changing microinstructions.
- **Maximizing** the simultaneous execution of microoperations.

Introduction to Computer Systems

Simple computer model: horizontal design of instruction - LOAD



	DO†	RW	MR	LD	AD	LA	WR	SR$_1$	SR$_0$	ST	Next address	
0	0	0	0	0	0	0	0	0	0	1	000	If START = 1 **then goto** 1 **else goto** 0
1	0	0	0	0	0	1	0	0	0	0	010	MAR <= IR (3 **downto** 0)
2	0	0	0	0	1	0	0	0	0	0	011	ADDR <= MAR
3	0	0	1	0	0	0	0	0	0	0	100	MR <= 1 and RW <= 0
4	0	0	1	1	0	0	0	0	0	0	101	MDR <= DATA
5	0	0	0	0	0	0	1	1	0	0	110	MR <= 0 and SR$_1$ <= 1 and SR$_0$ <= 0 and WR <= 1 reg_file (IR (5 **downto** 4)) <= MDR
6	1	0	0	0	0	0	0	0	0	0	000	decode_opfetch <= false

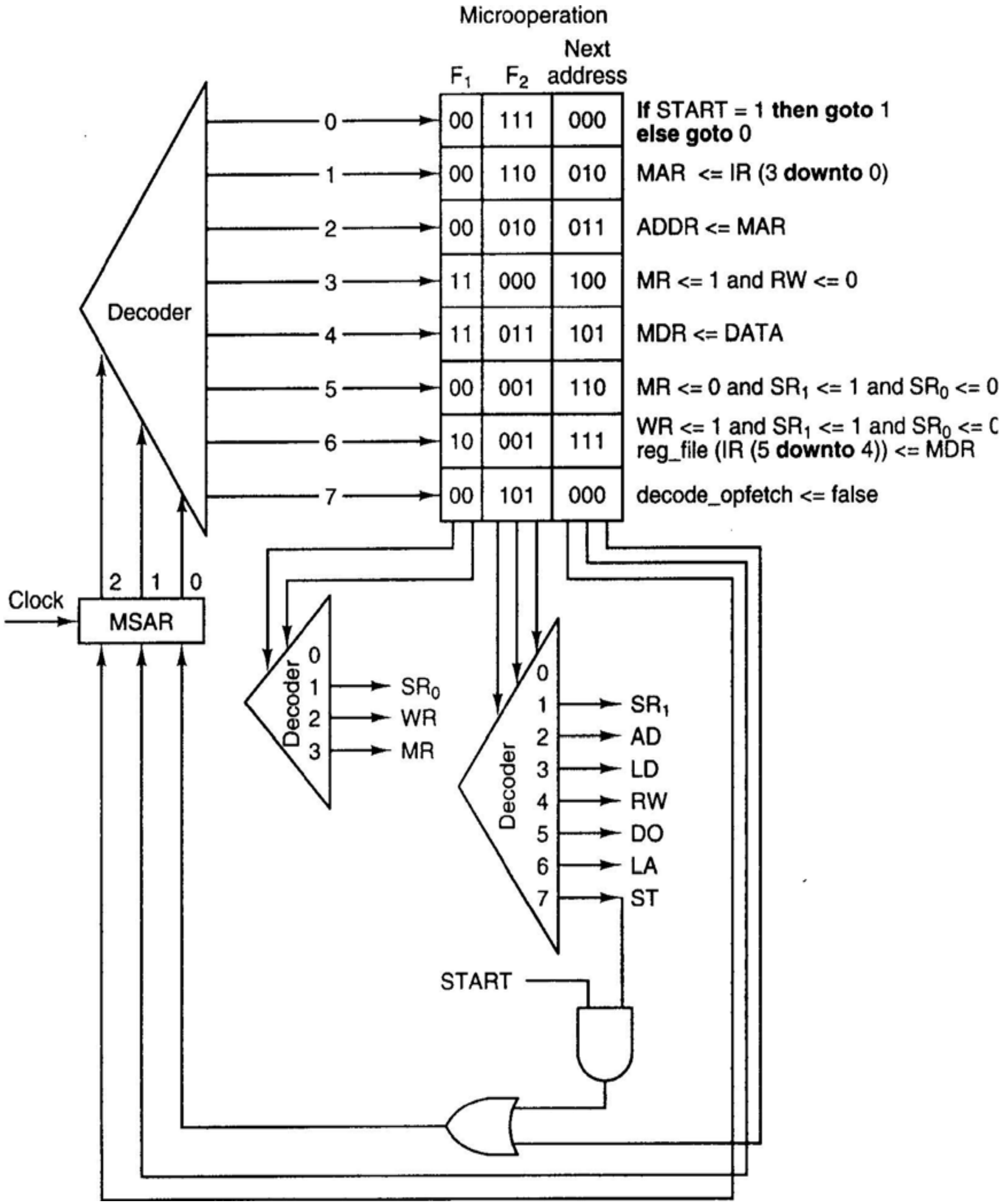
Advantage: Great flexibility, shorter execution time

Disadvantage: Wider microword size

Dangerous: you can activate multiple control signals at the same time, which will possibly damage the CPU.

Introduction to Computer Systems

Simple computer model: vertical design of instruction - LOAD



	Microoperation			
	F_1	F_2	Next address	
0	00	111	000	**If** START = 1 **then goto** 1 **else goto** 0
1	00	110	010	MAR <= IR (3 **downto** 0)
2	00	010	011	ADDR <= MAR
3	11	000	100	MR <= 1 and RW <= 0
4	11	011	101	MDR <= DATA
5	00	001	110	MR <= 0 and SR_1 <= 1 and SR_0 <= 0
6	10	001	111	WR <= 1 and SR_1 <= 1 and SR_0 <= 0 reg_file (IR (5 **downto** 4)) <= MDR
7	00	101	000	decode_opfetch <= false

Disadvantage: worse flexibility, longer execution time

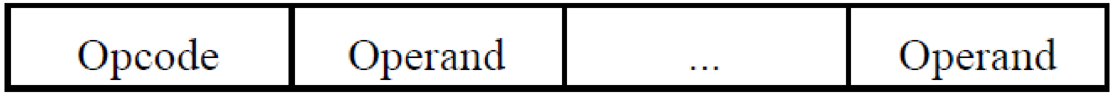
Disadvantage: smaller microword size

Here you can group control signals that must not be activated at the same time through different decoders

Introduction to Computer Systems

Simple computer model: instruction set design

The construction of an instruction set is one of the most important aspects of processor design.



Opcode	Operand	...	Operand

When an instruction set is designed, then one must seek answers to these questions.

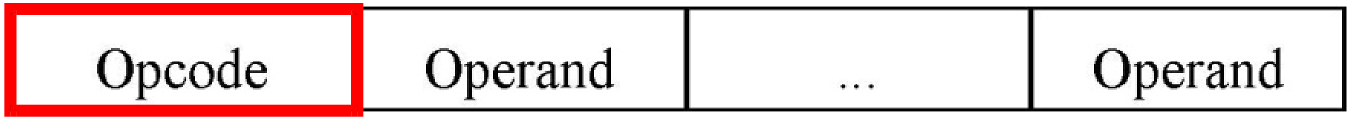
- How many instructions are needed?
- What types of operations are needed?
- How many operand fields and what types of operands will be allowed in each instruction?

Introduction to Computer Systems

Simple computer model: instruction set design

- How many instructions are needed?

The number of instructions helps determine the size of the Opcode



Opcode | Operand | … | Operand

The larger the number of instructions, the wider the Opcode field.
Wider Opcode field →each instruction takes up more memory
More instructions →the program can be written with fewer and more precise instructions

More instructions lead to the **CISC** design:
Complex **I**nstruction **S**et **C**omputer

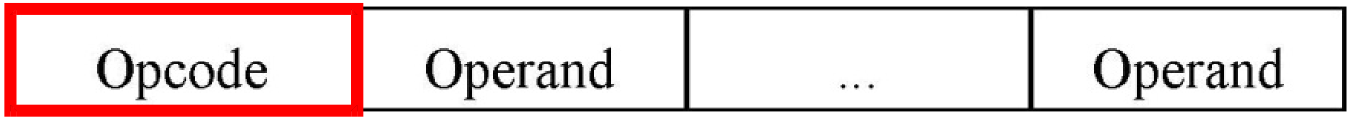
The complexity means that each instruction can take several clock cycles, but the total execution speed of a program is increased due to less instructions.

Introduction to Computer Systems

Simple computer model: instruction set design

- How many instructions are needed?

The number of instructions helps determine the size of the Opcode



The fewer the number of instructions, the narrower the Opcode field.
Narrower Opcode field →each instruction takes up less memory
Fewer instructions →the program takes up more space as more Instructions need to be done to achieve the same functionality

Fewer instructions lead to the **RISC** design:
Reduced **I**nstruction **S**et **C**omputer

Simplicity means that instructions are often performed within 1 clock cycle, so the execution speed of individual instruction is increased.

Introduction to Computer Systems

Simple computer model: instruction set design

- How many instructions are needed?

This is a trade-off of whether you want functionality to be implemented as instructions in the CPU:

- Shorter access time (compared to access to memory)
- However, slower than RISC due to the complexity
- Smaller program
- More power consumption

Or as subroutines in the program:

- Longer access time (compared to access to the CPU)
- However, faster than CISC thanks to the simplicity
- Larger program (with subroutines or libraries)
- Less power consumption

Introduction to Computer Systems

Simple computer model: instruction set design

- What types of operations are needed?

Although the instruction sets vary in the different processor types, they almost all contain the same types of operations. These operations can be divided as follows:

- **Data transfer operations**: move (or copying) data from one location (memory or register) to another location (also memory or register).
- **Arithmetic operations**: perform basic arithmetic operations.
- **Logical operations**: perform Boolean operations.
- **Control operations**: check a sequence of instruction executions, e.g., SKIP, RETURN, HALT.
- **System operations**: Instructions specifically designed for use with the operating system, e.g., system calls and memory management.
- **I/O operations**: move data between memory and external I/O devices.

Introduction to Computer Systems

Simple computer model: instruction set design

- How many operand fields and what types of operands will be allowed in each instruction?

In a typical set of instructions, the size of the operand field is quite limited. However, it is necessary to be able to refer to a very large amount of memory from this limited operand field. In order to achieve this, various **addressing modes** have been introduced. The most commonly used addressing modes are:

- *Immediate addressing*
- *Direct addressing*
- *Indirect addressing*
- *Displacement addressing*
- *Stack addressing*

Introduction to Computer Systems

Simple computer model: instruction set design

Immediate addressing

In this addressing mode, the operand field contains the data itself and the data does not have to be retrieved from the memory first via an operand fetch.

The disadvantage is that the value range of the operands is limited by the limited size of the operand field and that data must be known at the time of compilation (constants).

Introduction to Computer Systems

Simple computer model: instruction set design

Direct addressing

In this addressing mode, the operand field contains an address for the data you want to access, which is also known as **absolute addressing**

If the field points to a register that contains data, this is called **register direct addressing.**

It only requires a memory (or register) reference, which does not require any special calculation of the address in order to perform an operand fetch. However, the size of the operand field has a direct influence on the size of memory that can be addressed.

Introduction to Computer Systems

Simple computer model: instruction set design

Indirect addressing

In this addressing mode, the operand field contains an address of a memory location or a register for the target address, which contains the data you want to access.

If the field points to a register which contains the address of the data you want to access, this is called **register indirect addressing.**

In this addressing mode, the area that can be addressed depends on the length of the memory cell or register referenced via the operand field. It will often be a much larger area than what can be addressed with direct addressing mode. The main disadvantage is that two references are required to reach the target operand.

Introduction to Computer Systems

Simple computer model: instruction set design

Displacement addressing

In this addressing mode, direct addressing and register indirect addressing are combined. It requires that the operand field consists of two subfields. One of the two subfield is used to provide an address to access to a predefined register whose value is added to the value of the other subfield. The sum is the target address.

The value of one operand field is called the offset from the address the other operand field points to. The term displacement is used because the values in the subfields are not large enough to address the entire memory.

Introduction to Computer Systems

Simple computer model: instruction set design

Stack addressing

A stack can be considered as a linear array of address locations. Data units can be only moved to or removed from the top of a stack.

The stack addresses are actually one kind of **register indirect addresses**, since the address of the top element of a stack is always stored by a special register, namely the **stack register**.

This addressing mode is used, for example, by **Java**, as it does not make any special demands on the hardware.

On the other hand, it is not very fast, as memory is accessed frequently.