Computer system basics

- A digital hardware: a lot of switches integrated
- A digital switch: the electronic device react to presence or absence of voltage
- Symbolically we represent
  - Presence of voltage as “1”
  - Absence of voltage as “0”

Computer system basics cont.

- An electronic device can represent uniquely only one of two things
  - Each “0” or “1” is referred to as a Binary Digit or Bit
  - Bit: Fundament unit of information storage
- To represent more things we need more bits
  - E.g., 2 bits can represent four unique things: 00, 01, 10,11
  - k bits can distinguish 2^k distinct items
- Combination binary bits together can represent some info. or data. E.g., 01000001 can be
  1. Decimal value 65
  2. Alphabet (or character) ‘**A**’ in ASCII notation
  3. Command to be performed, e.g., performing **Add** opration

We can divide a computer into three broad parts or subsystems:

1. Central Processing Unit (CPU),
2. main memory and
3. input/output subsystem.

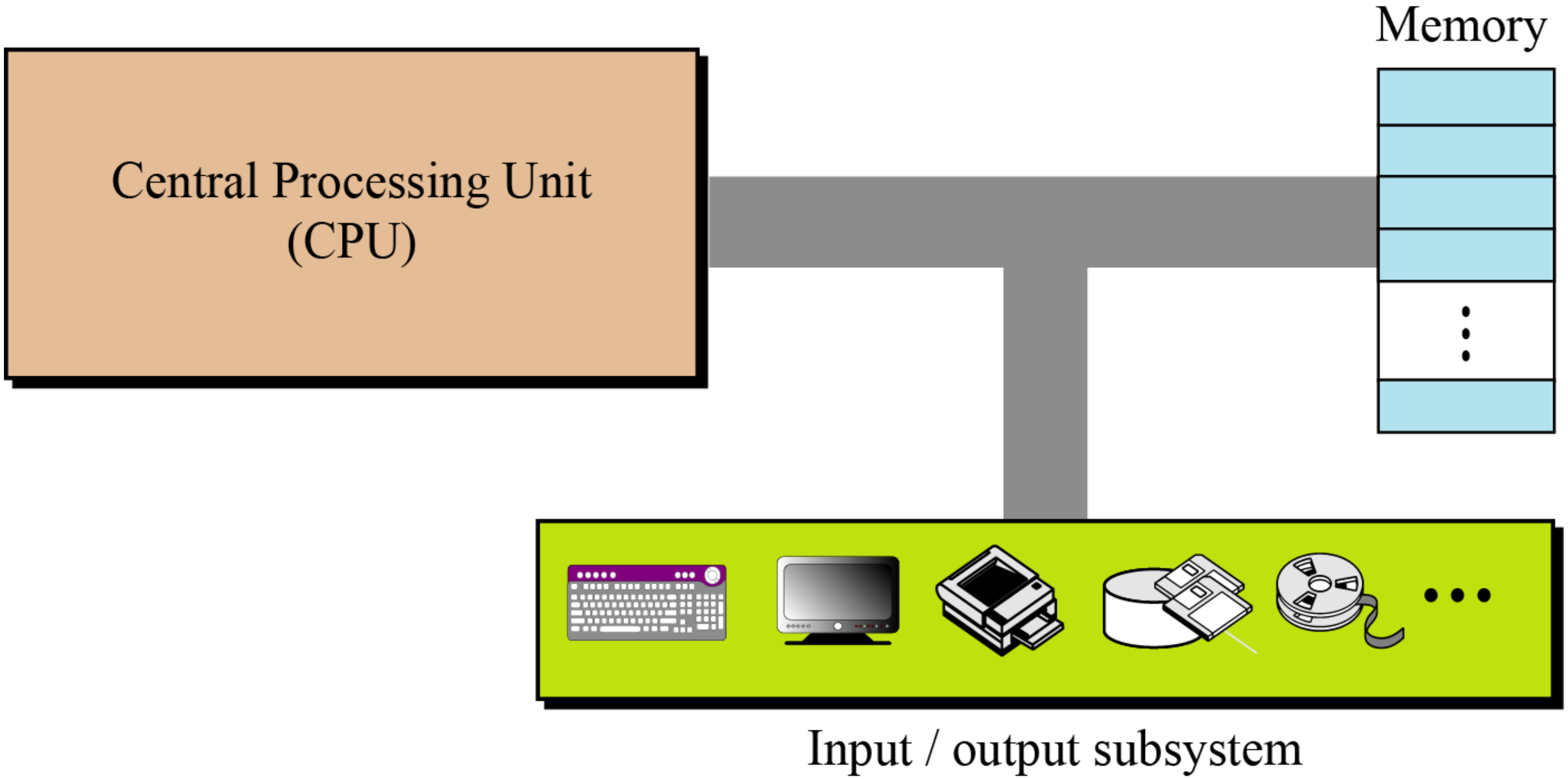


Figure: Computer hardware (subsystems)

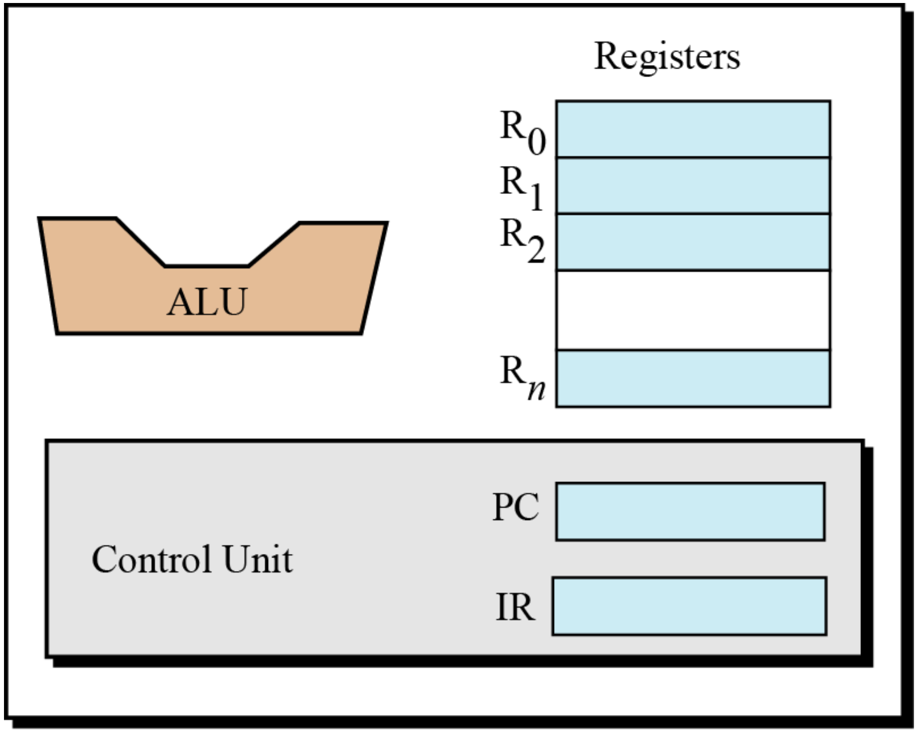
Central Processing Unit (CPU)

The CPU performs operations on data. In most architectures it has three parts:

1. an arithmetic logic unit (ALU),

2. a control unit and

3. a set of registers, fast storage



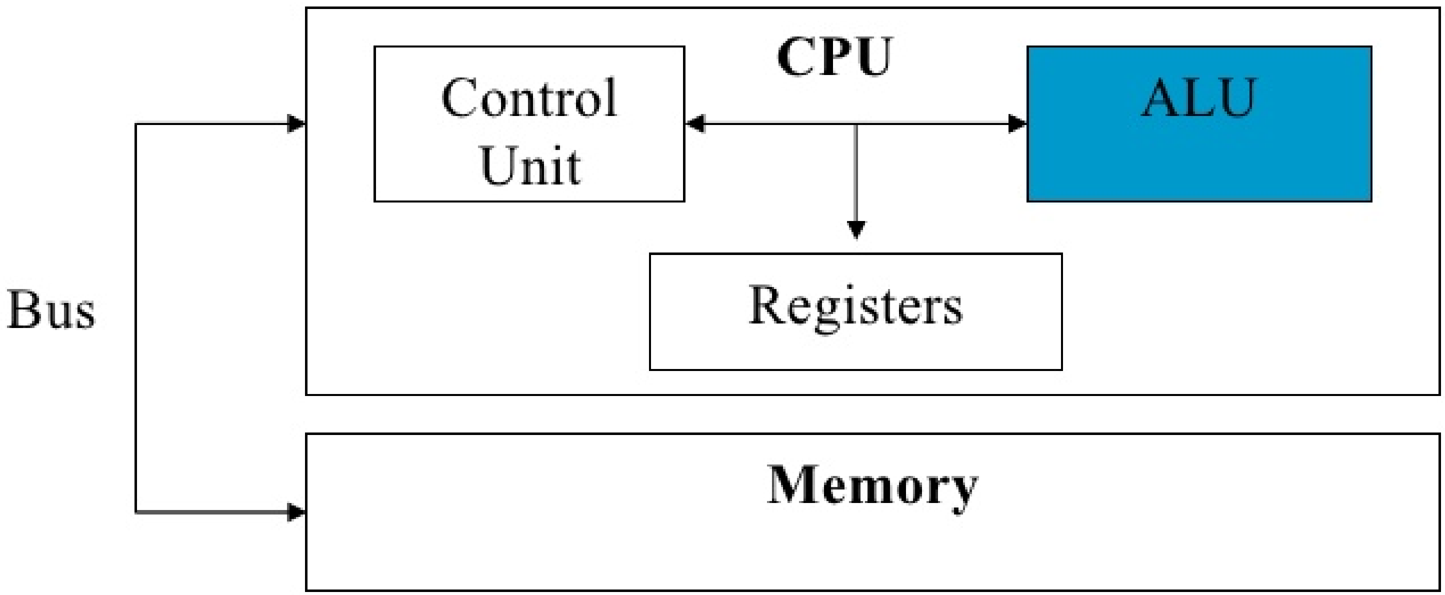
Central Processing Unit (CPU)

PC: Program counter

IR: Instruction register

ALU

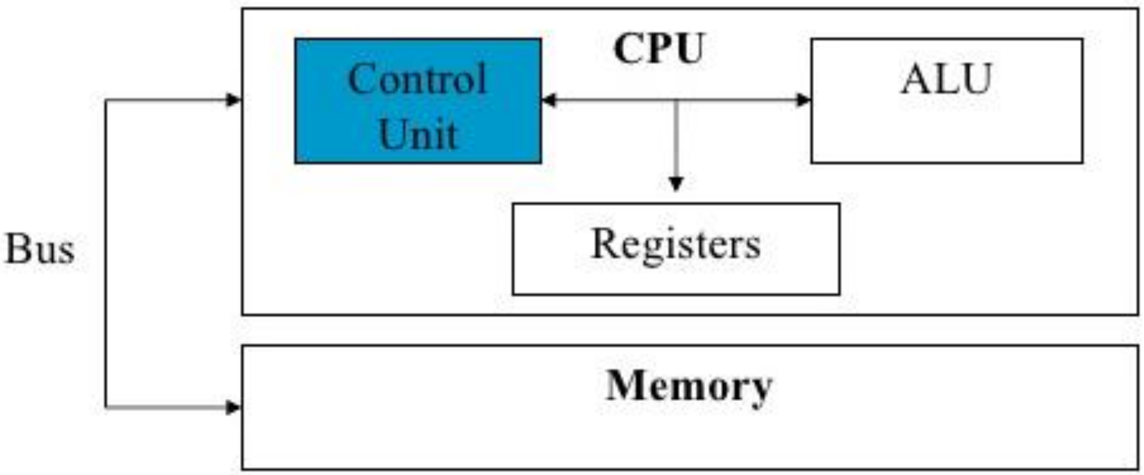
Computer System



- **Arithmetic and Logic Unit (ALU)**
  - It performs calculations and comparisons of data.

Control Unit

Computer System



Control Unit

Note: The control unit controls the computer by repeating 4 operations, called the machine cycle. The 4 operations are: fetching program instructions from memory; decoding the instructions into commands that the computer can process; executing the commands; and storing the results in memory

Registers

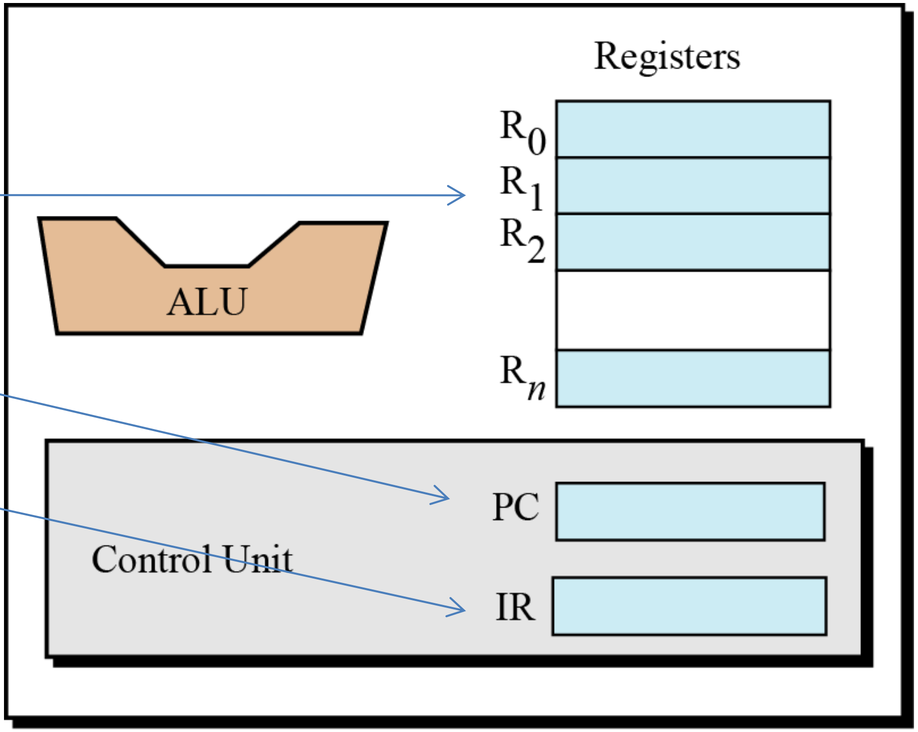
Registers are fast stand-alone storage locations that hold data temporarily.

Multiple registers are needed to facilitate the operation of the CPU.

❑ Data registers

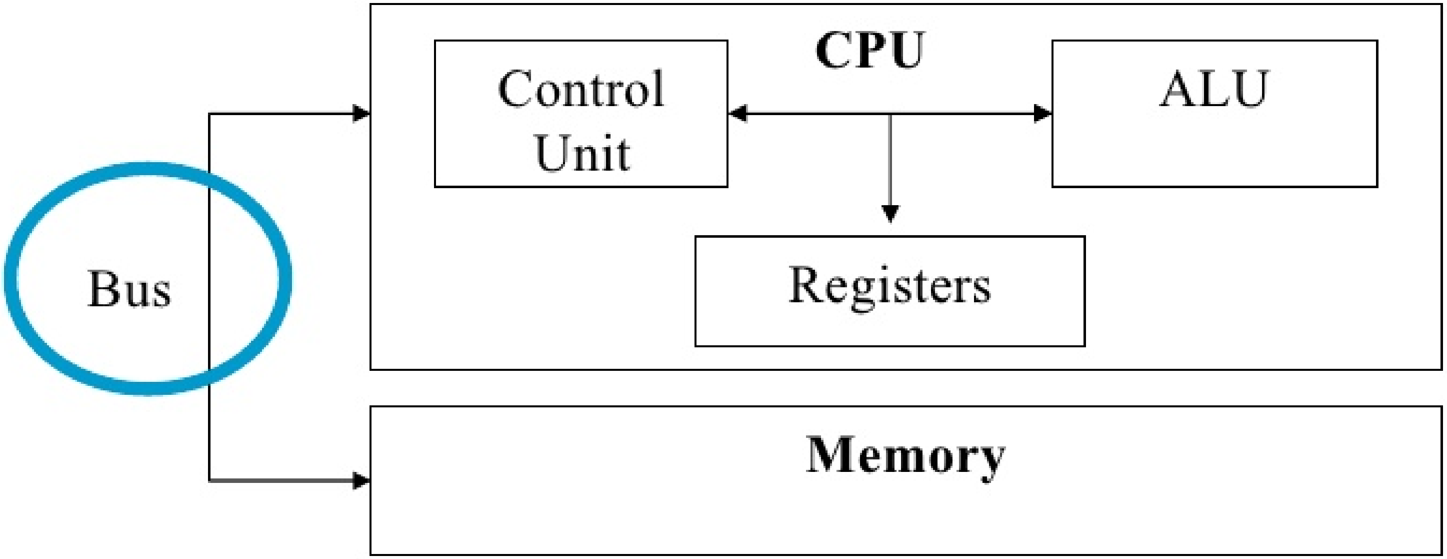
❑ Program counter

❑ Instruction register



Central Processing Unit (CPU)

Computer System



Buses

They are electrical pathways that carry signal (bits) between a CPU's components and outside devices.

MAIN MEMORY

accepts and holds program instruction and data

⇨ acts as the CPU’s source for data and instructions and as a destination for operation results

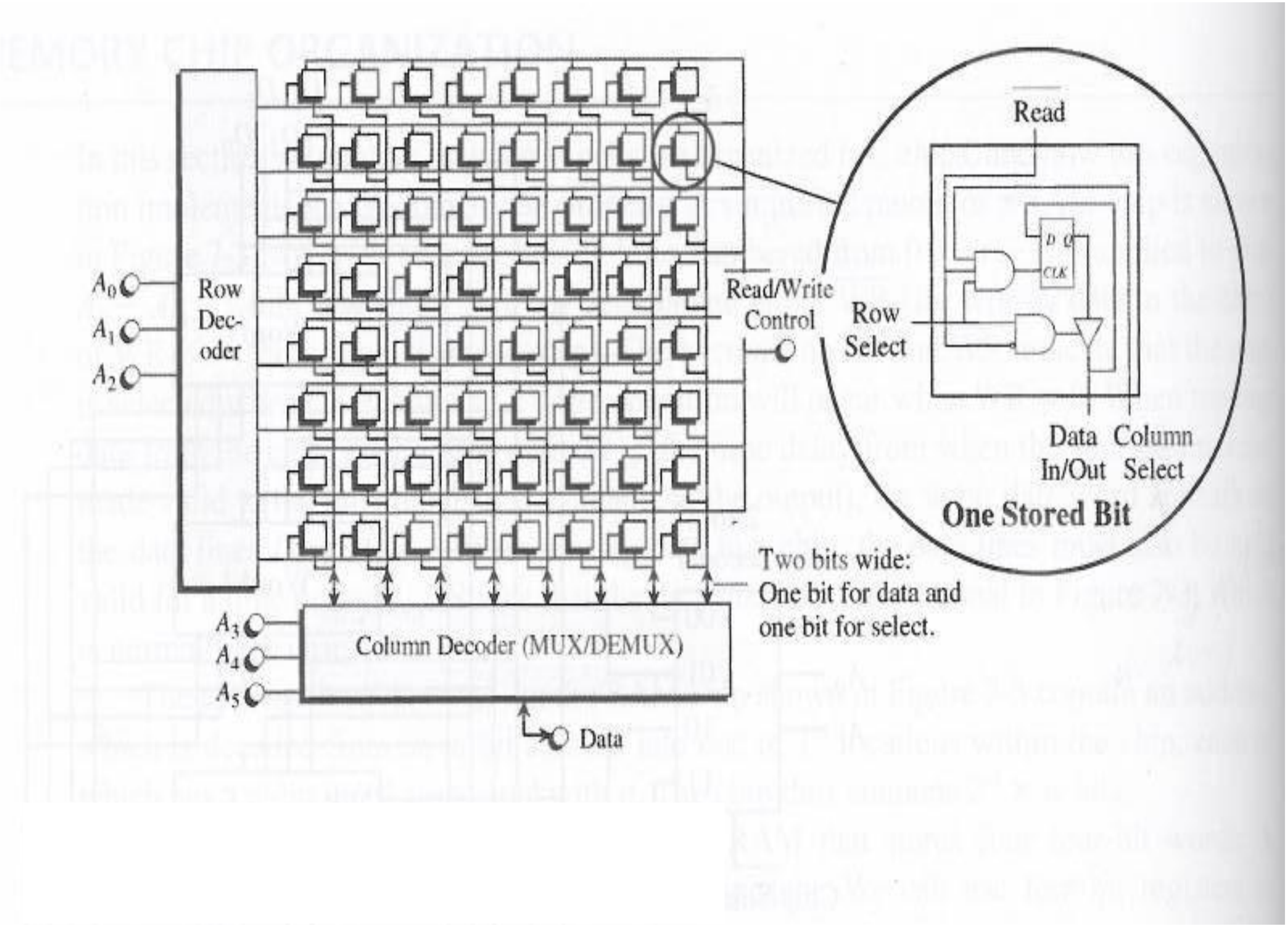
⇨ holds the final processed information until it can be sent to the desired output or storage devices, such as printer or disk drive

MAIN MEMORY

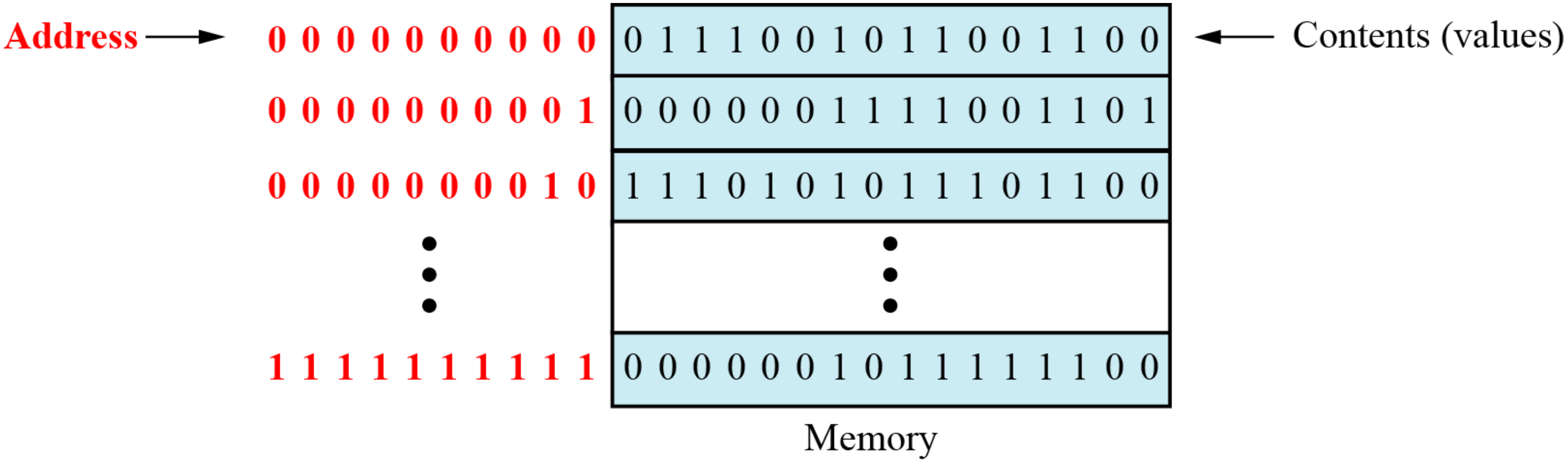
Main memory consists of a collection of storage locations, each with a unique id, called an address.

Data is transferred to and from memory in groups of bits called words. A word can be a group of 8 bits, 16 bits, 32 bits or 64 bits (and growing). If the word is 8 bits, it is referred to as a byte.

RAM Grid



A_0
A_1
A_2
Row Dec-oder
Read/Write Control
Read
D Q
CLK
Row Select
Data In/Out
Column Select
One Stored Bit
Two bits wide: One bit for data and one bit for select.
A_3
A_4
A_5
Column Decoder (MUX/DEMUX)
Data



0 0 0 0 0 0 0 0 0 0	0 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0
0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 1 1 1 1 0 0 1 1 0 1
0 0 0 0 0 0 0 0 1 0	1 1 1 0 1 0 1 0 1 1 1 0 1 1 0 0
⋮	⋮
1 1 1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 0

Figure: Main memory

Address space

To access a word in memory requires an identifier. Although programmers use a name to identify a word (or a collection of words), at the hardware level each word is identified by an address.

The total number of uniquely identifiable locations in memory is called the address space.
For example, a memory with 64 kilobytes and a word size of 1 byte has an address space that ranges from 0 to 65,535.

a byte-addressable 32-bit computer can address
2^{32} = 4,294,967,296 bytes of memory, or 4 gibibytes (GiB)

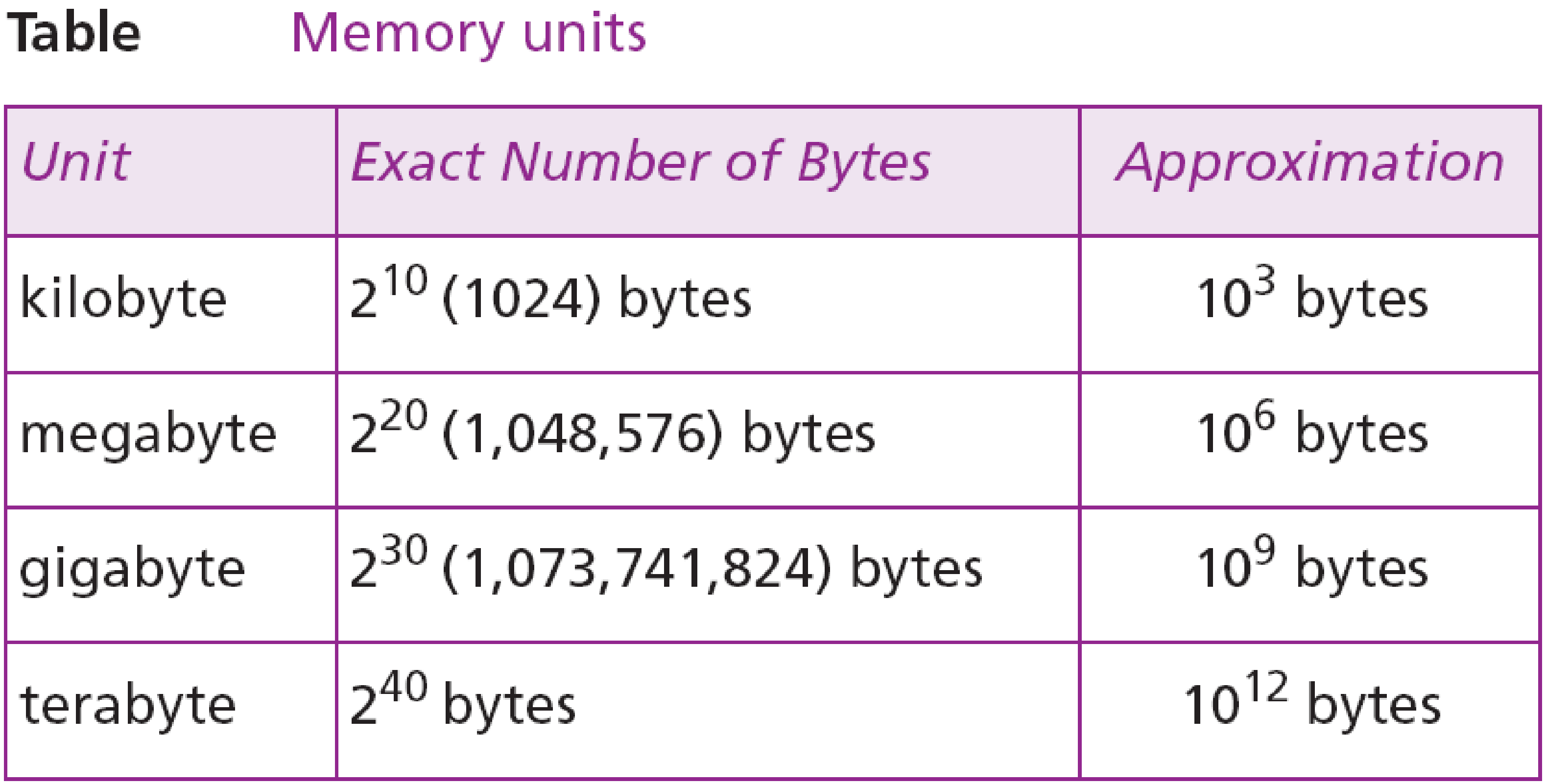


Table Memory units

Unit	*Exact Number of Bytes*	*Approximation*
kilobyte	2^{10} (1024) bytes	10^3 bytes
megabyte	2^{20} (1,048,576) bytes	10^6 bytes
gigabyte	2^{30} (1,073,741,824) bytes	10^9 bytes
terabyte	2^{40} bytes	10^{12} bytes

Memory addresses are defined using unsigned binary integers

Example 1

A computer has 32 MB (megabytes) of memory. How many bits are needed to address any single byte in memory?

Solution

The memory address space is 32 MB= 2^{25} ($2^5 \times 2^{20}$). This means that we need $\log_2 2^{25}$=25 bits, to address each byte.

Example 2

A computer has 128 MB of memory. Each word in this computer is eight bytes. How many bits are needed to address any single word in memory?

Solution

The memory address space is 128 MB, which means 2^{27}. However, each word is eight (2^3) bytes, which means that we have 2^{24} words. This means that we need $\log_2 2^{24}$, or 24 bits, to address each word.

A SIMPLE COMPUTER

To explain the architecture of computers as well as their instruction processing, we introduce a simple (unrealistic) computer shown in next Figure.

Our simple computer has three components: CPU, main memory and an input/output subsystem.

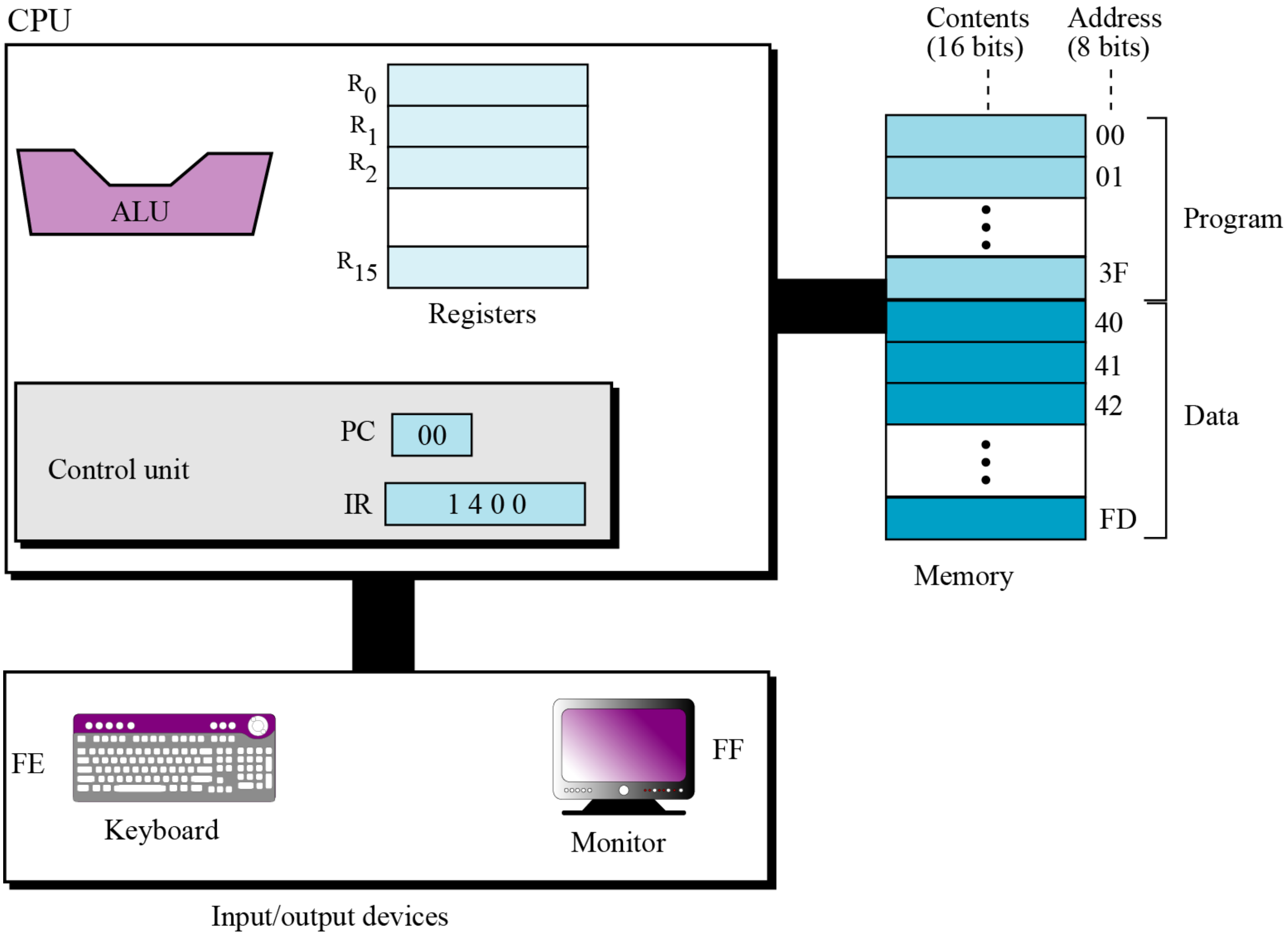


Figure: The components of a simple computer

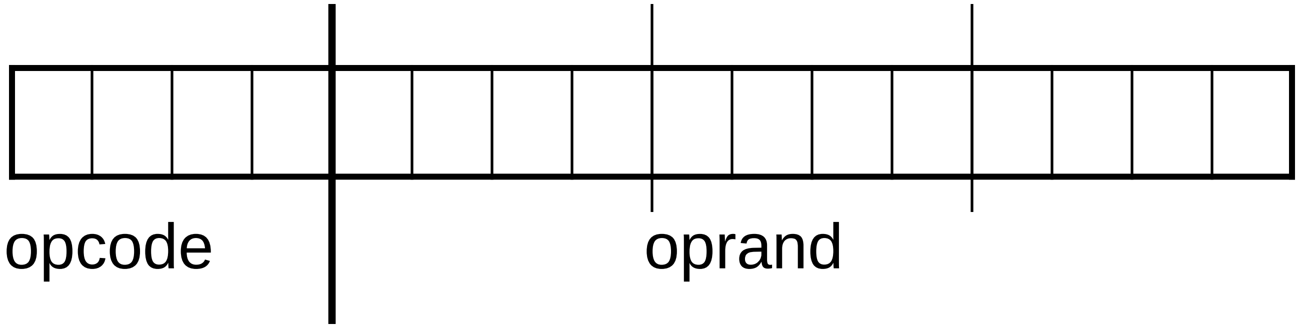
Instruction set

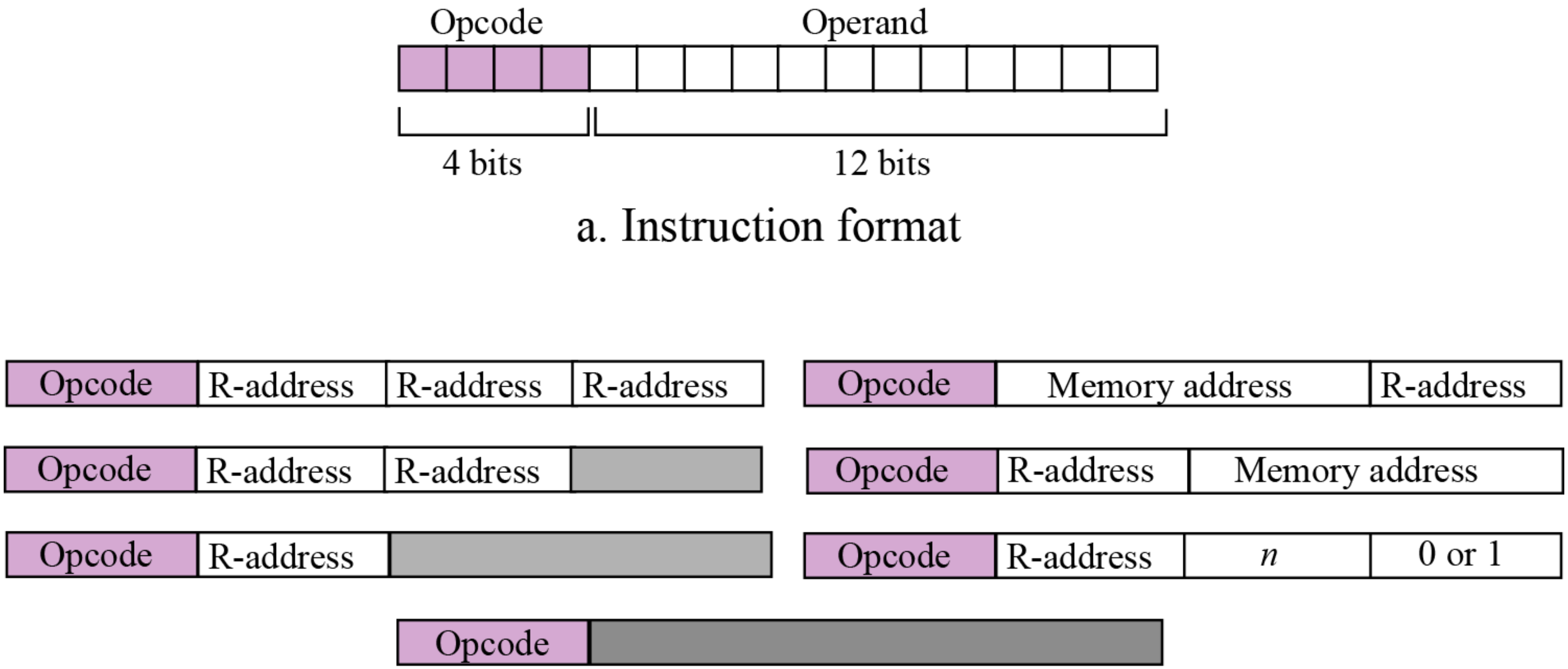
Each computer instruction consists of two parts: the operation code (opcode) and the operand (s).

The opcode specifies the type of operation to be performed on the operand (s).

Each instruction consists of sixteen bits divided into four 4-bit fields.

The leftmost field contains the opcode and the other three fields contains the operand or address of operand (s).




Opcode
Operand
4 bits
12 bits
Opcode | R-address | R-address | R-address
Opcode | R-address | R-address
Opcode | R-address
Opcode
Opcode | Memory address | R-address
Opcode | R-address | Memory address
Opcode | R-address | n | 0 or 1

a. Instruction format

b. Instruction types

Figure: Format and different instruction types

R-address: address of a register

Memory address: Main memory address

Processing the instructions

Our simple computer, like most computers, uses machine cycles.

A cycle is made of three phases: fetch, decode and execute.

During the fetch phase, the instruction whose address is in PC is obtained from the memory and loaded into the IR. The PC is then incremented to point to the next instruction.

During the decode phase, the instruction in IR is decoded and the required operands are fetched from the register or from memory.

During the execute phase, the instruction is executed and the results are placed in the appropriate memory location or the register.

Once the third phase is completed, the control unit starts the cycle again, but now the PC is pointing to the next instruction. The process continues until the CPU reaches a HALT instruction.

Table List of instructions for the simple computer

Instruction	*Code*	*Operands*			*Action*
	d_1	d_2	d_3	d_4	
HALT	0				Stops the execution of the program
LOAD	1	R_D	M_S		$R_D \leftarrow M_S$
STORE	2	M_D		R_S	$M_D \leftarrow R_S$
ADDI	3	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
ADDF	4	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
MOVE	5	R_D	R_S		$R_D \leftarrow R_S$
NOT	6	R_D	R_S		$R_D \leftarrow \overline{R_S}$
AND	7	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ AND R_{S2}
OR	8	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ OR R_{S2}
XOR	9	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ XOR R_{S2}
INC	A	R			$R \leftarrow R + 1$
DEC	B	R			$R \leftarrow R - 1$
ROTATE	C	R	n	0 or 1	Rot_n R
JUMP	D	R	n		IF $R_0 \neq R$ then PC = *n*, otherwise continue

Key: R_S, R_{S1}, R_{S2}: Hexadecimal address of source registers
R_D: Hexadecimal address of destination register
M_S: Hexadecimal address of source memory location
M_D: Hexadecimal address of destination memory location
n: hexadecimal number
d_1, d_2, d_3, d_4: First, second, third, and fourth hexadecimal digits

An example

Let us show how our simple computer can add two integers A and B and create the result as C.

We assume that integers are in two’s complement format. Mathematically, we show this operation as:

$$C = A + B$$

We assume that the first two integers are stored in memory locations $(40)_{16}$ and $(41)_{16}$ and
the result should be stored in memory location $(42)_{16}$.

To do the simple addition needs five instructions, as shown next:

1. Load the contents of M_{40} into register R_0 ($R_0 \leftarrow M_{40}$).
2. Load the contents of M_{41} into register R_1 ($R_1 \leftarrow M_{41}$).
3. Add the contents of R_0 and R_1 and place the result in R_2 ($R_2 \leftarrow R_0 + R_1$).
4. Store the contents R_2 in M_{42} ($M_{42} \leftarrow$ R2).
5. Halt.

In the language of our simple computer, these five instructions are encoded as:

Code	*Interpretation*			
$(1040)_{16}$	**1**: LOAD	0: R_0	40: M_{40}	
$(1141)_{16}$	**1**: LOAD	1: R_1	41: M_{41}	
$(3201)_{16}$	**3**: ADDI	2: R_2	0: R_0	1: R_1
$(2422)_{16}$	**2**: STORE	42: M_{42}		2: R_2
$(0000)_{16}$	**0**: HALT			

Table List of instructions for the simple computer

Instruction	*Code*	*Operands*			*Action*
	d_1	d_2	d_3	d_4	
HALT	0				Stops the execution of the program
LOAD	1	R_D	M_S		$R_D \leftarrow M_S$
STORE	2	M_D		R_S	$M_D \leftarrow R_S$
ADDI	3	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
ADDF	4	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1} + R_{S2}$
MOVE	5	R_D	R_S		$R_D \leftarrow R_S$
NOT	6	R_D	R_S		$R_D \leftarrow \overline{R_S}$
AND	7	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ AND R_{S2}
OR	8	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ OR R_{S2}
XOR	9	R_D	R_{S1}	R_{S2}	$R_D \leftarrow R_{S1}$ XOR R_{S2}
INC	A	R			$R \leftarrow R + 1$
DEC	B	R			$R \leftarrow R - 1$
ROTATE	C	R	n	0 or 1	$\mathrm{Rot}_n R$
JUMP	D	R	n		IF $R_0 \neq R$ then PC = n, otherwise continue

Key: R_S, R_{S1}, R_{S2}: Hexadecimal address of source registers
R_D: Hexadecimal address of destination register
M_S: Hexadecimal address of source memory location
M_D: Hexadecimal address of destination memory location
n: hexadecimal number
d_1, d_2, d_3, d_4: First, second, third, and fourth hexadecimal digits

Storing program and data

We can store the five-line program in memory starting from location $(00)_{16}$ to $(04)_{16}$.

We already know that the data needs to be stored in memory locations $(40)_{16}$, $(41)_{16}$, and $(42)_{16}$.

Cycles

Our computer uses one cycle per instruction. If we have a small program with five instructions, we need five cycles. We also know that each cycle is normally made up of three steps: fetch, decode, execute.

Assume for the moment that we need to add 161 + 254 = 415. The numbers are shown in memory in hexadecimal is, $(00A1)_{16}$, $(00FE)_{16}$, and $(019F)_{16}$.

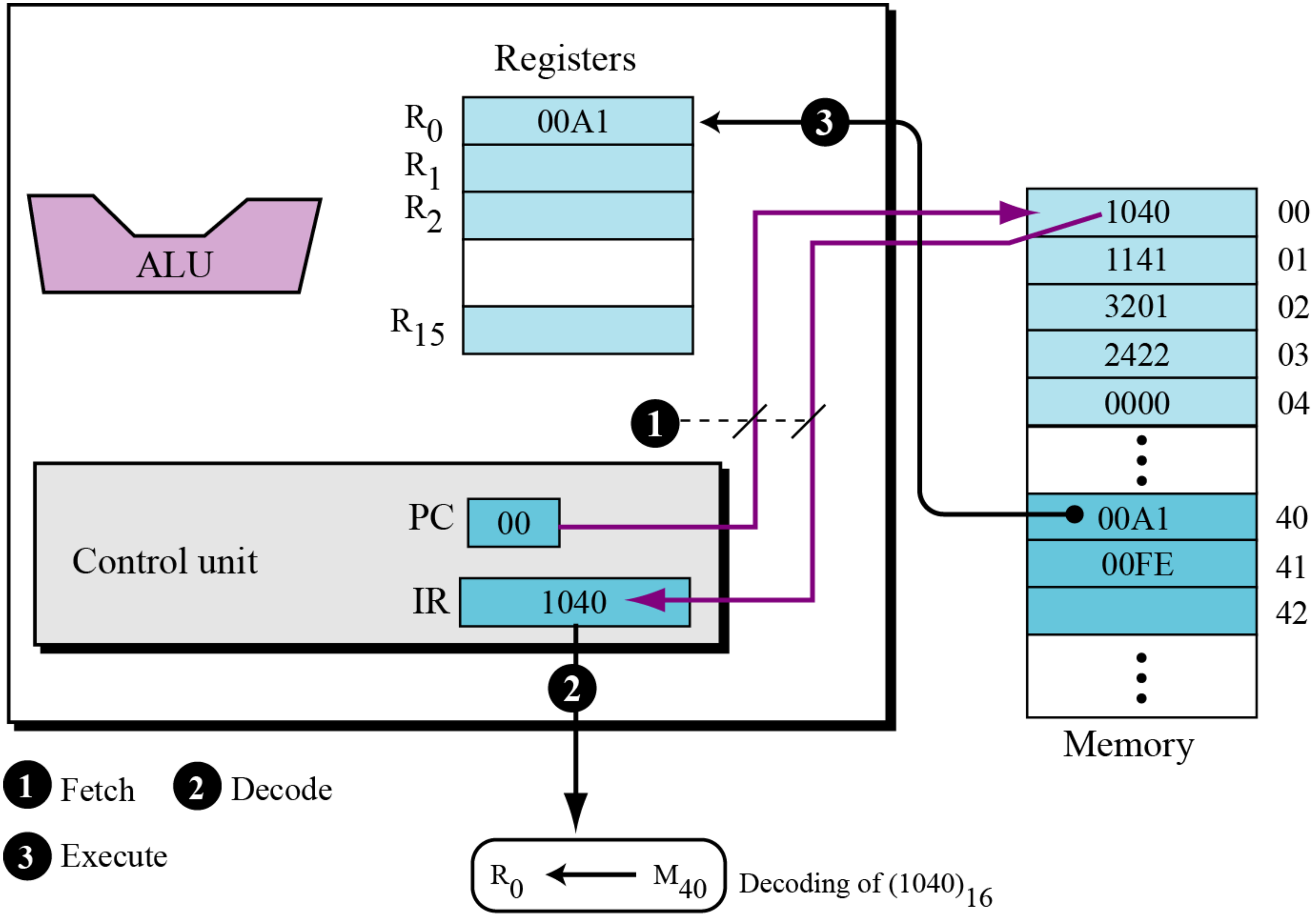


Figure Status of cycle 1

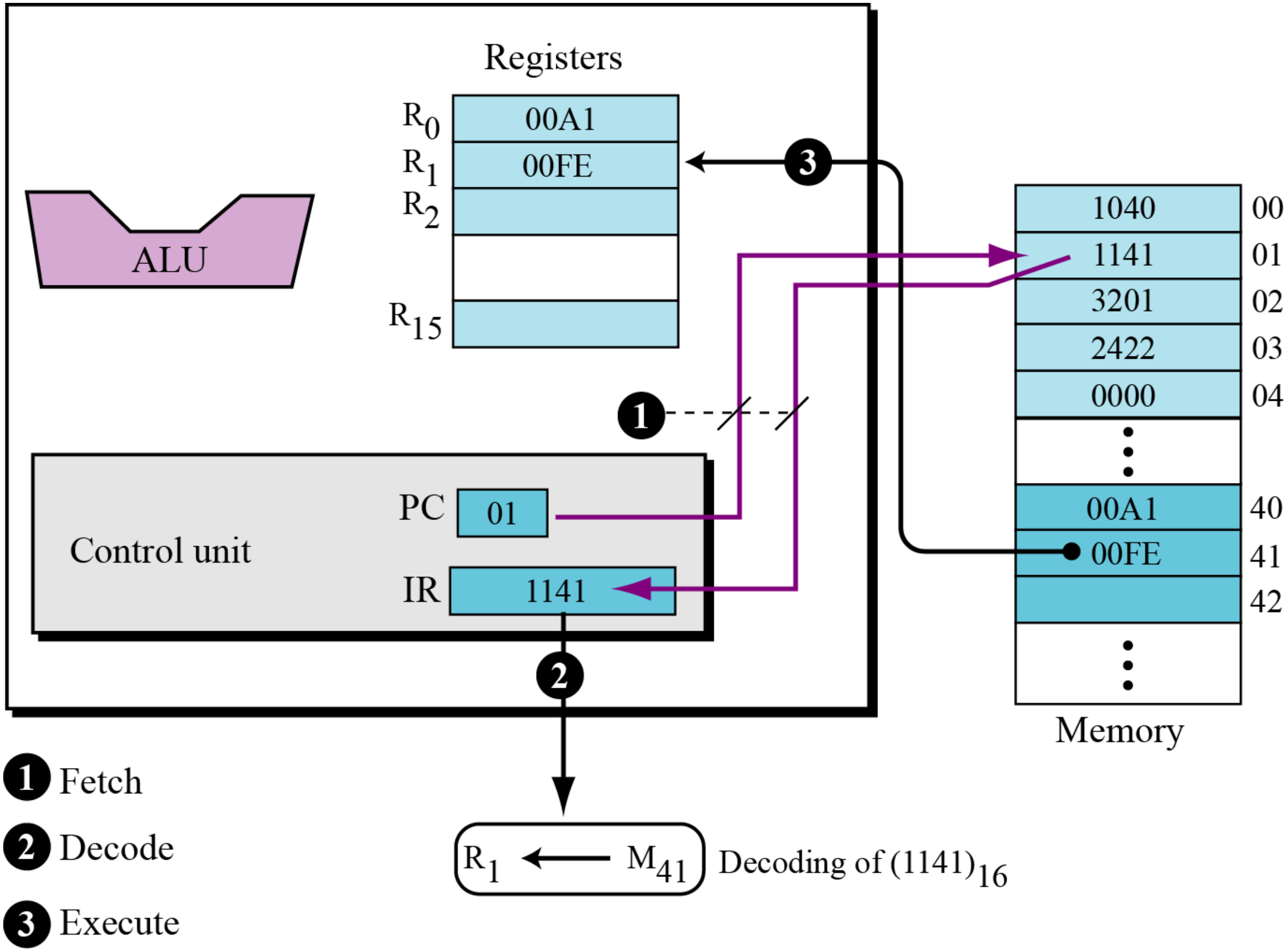


Figure Status of cycle 2

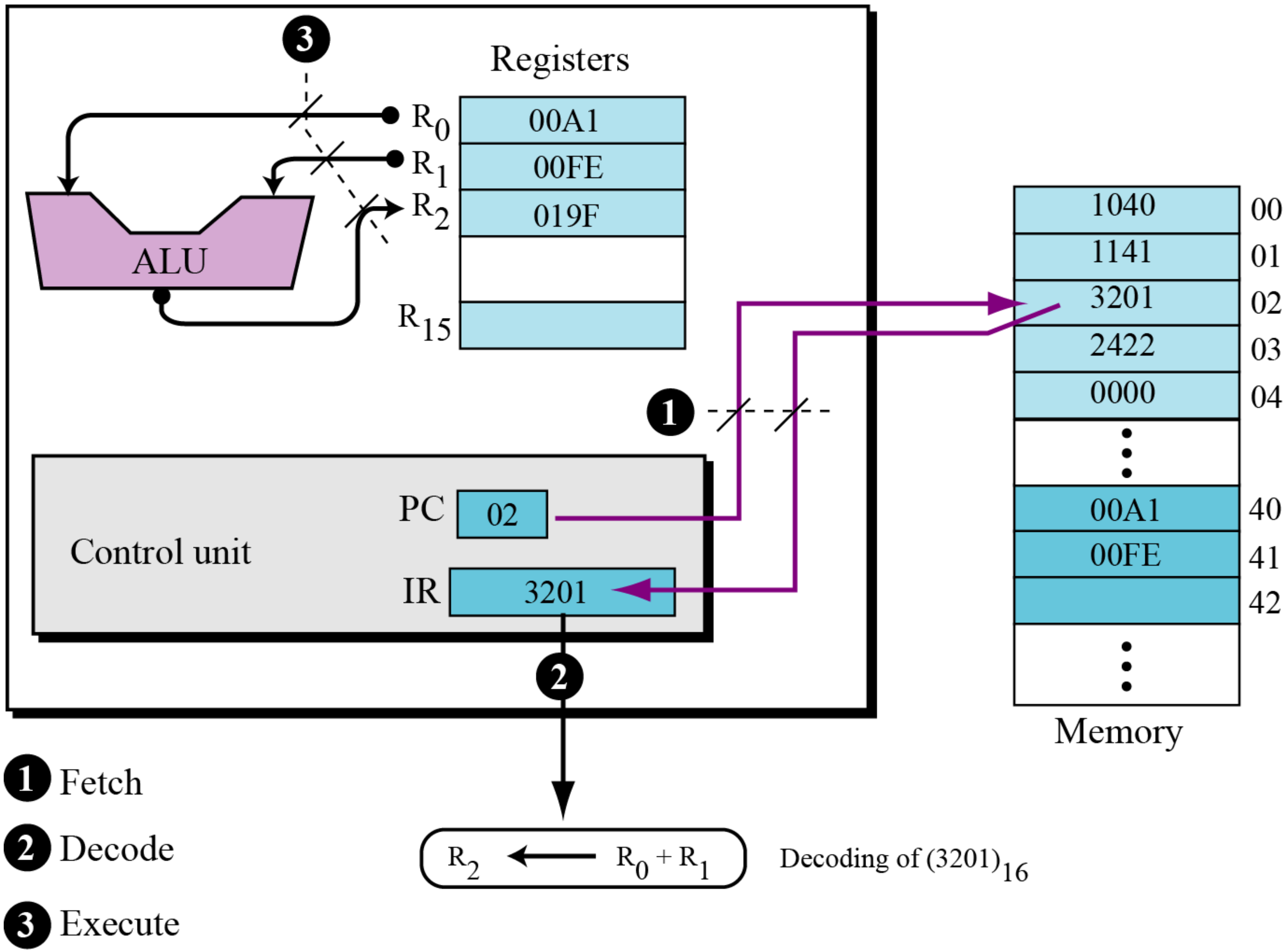


Figure: Status of cycle 3

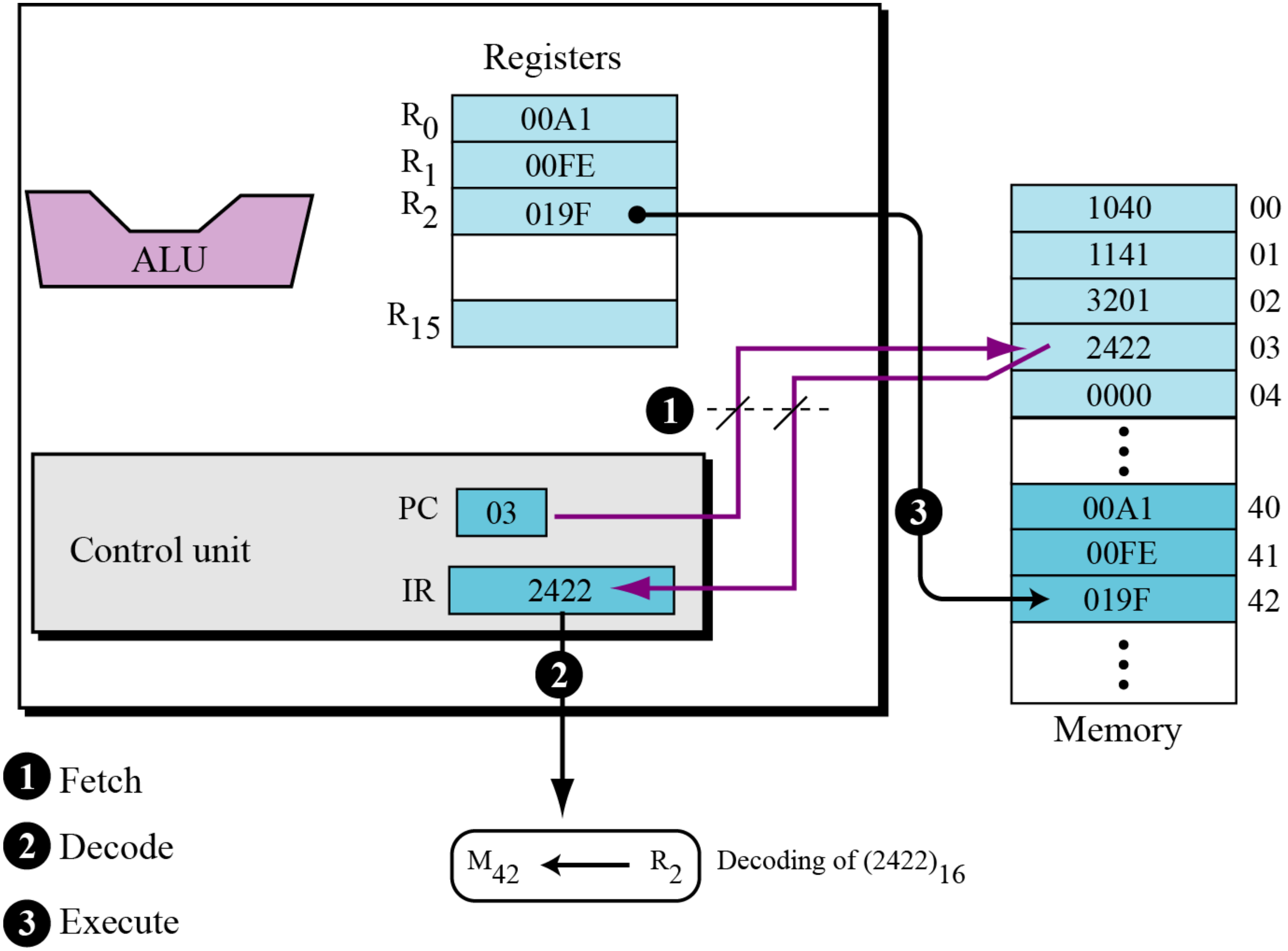


Figure: Status of cycle 4

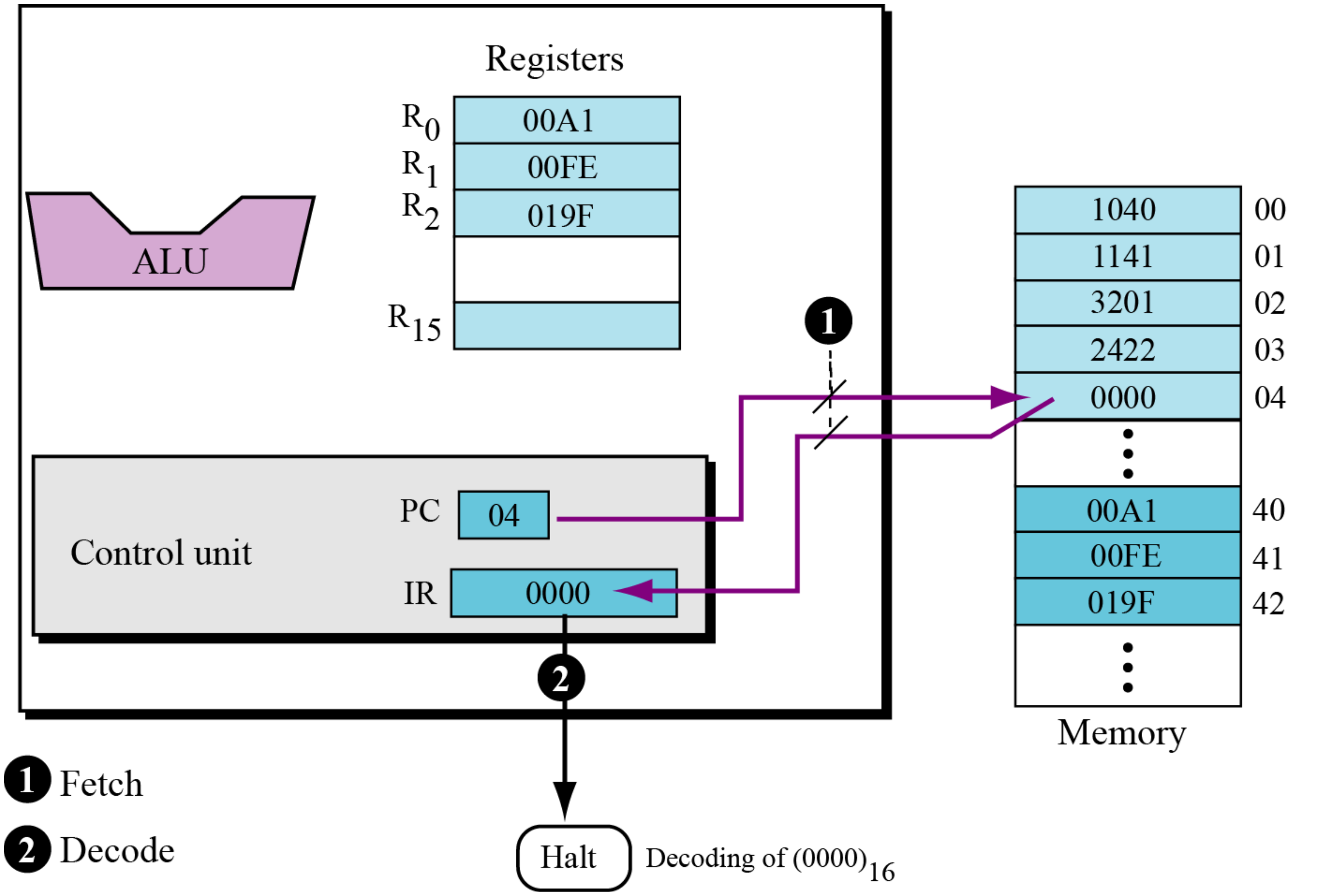


Figure: Status of cycle 5