Lecture 3

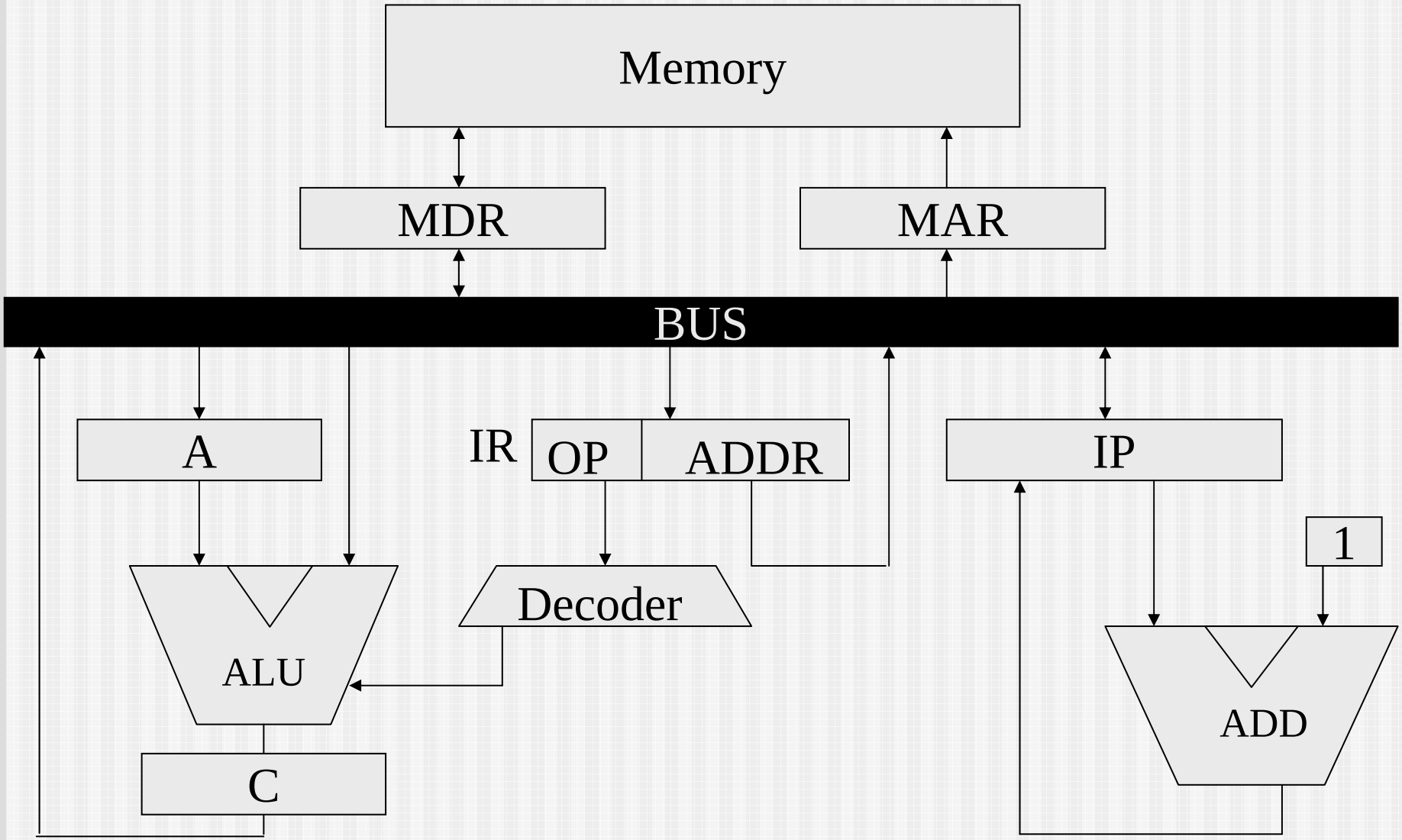
Computer Architecture

Bus Architectures

Bus(es)

- A bus is a hardware channel through which information can flow between components connected to the bus.
- It allows us to simplify our current model and to make further additions more easily.
- Only two components may communicate through a bus at any given time.

One Bus Design



Instruction Set

Store(01)

MDR<-BUS<-A

Memory[MAR]<- MDR

Load(02)

MDR <-Memory[MAR]

A <-BUS<-MDR

ADD(03)

MDR <-Memory[MAR]

C <- A + MDR *(ALU<-BUS<-MDR)*

A<-BUS<- C

END(04)

Stop

JMP (05)

PC ← IR.ADDR

Fetch(00)

MAR <- BUS<-PC

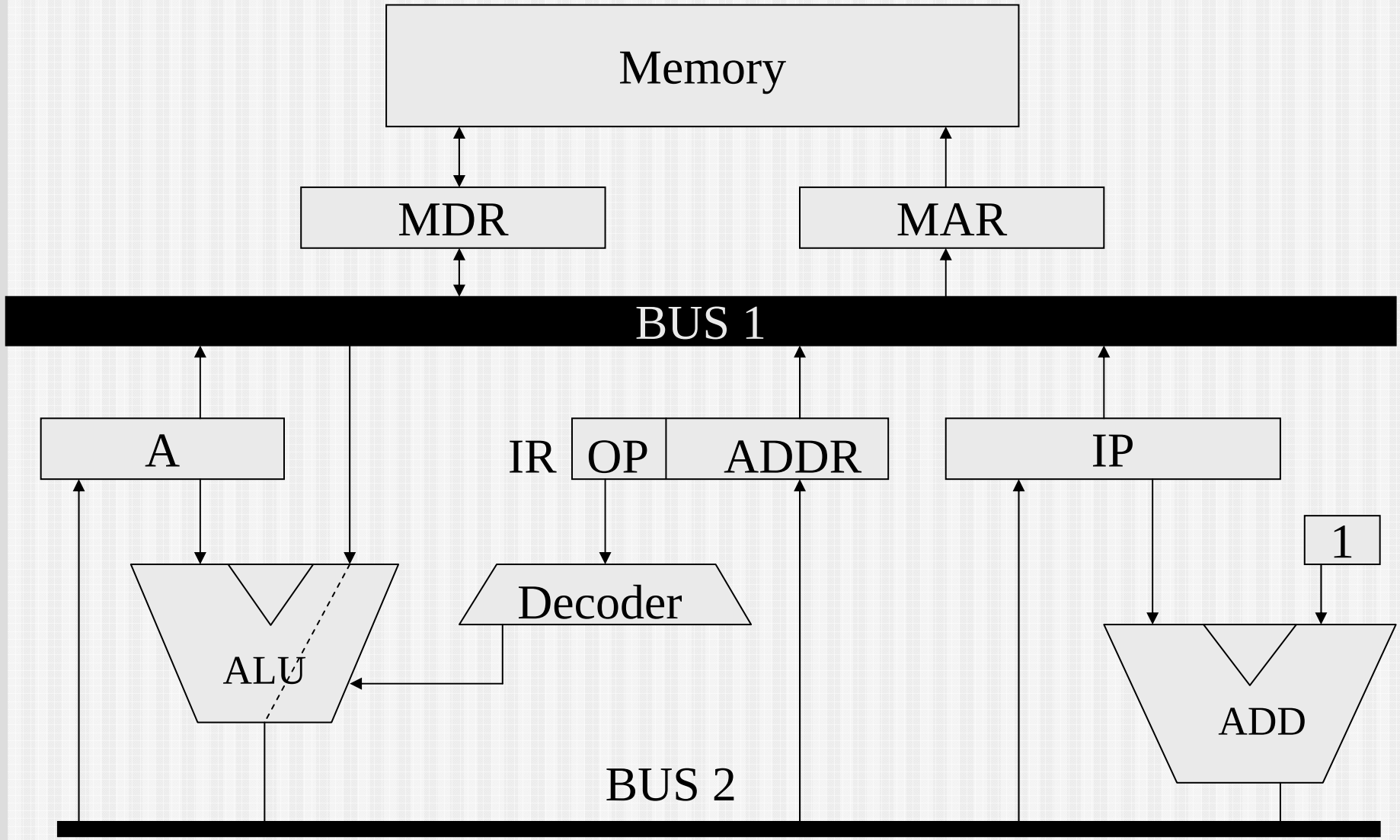
MDR <- Memory[MAR] || PC <-PC + 1

IR <-BUS<-MDR

Decoder <- IR.Opcode

MAR<-BUS<-IR.ADDR

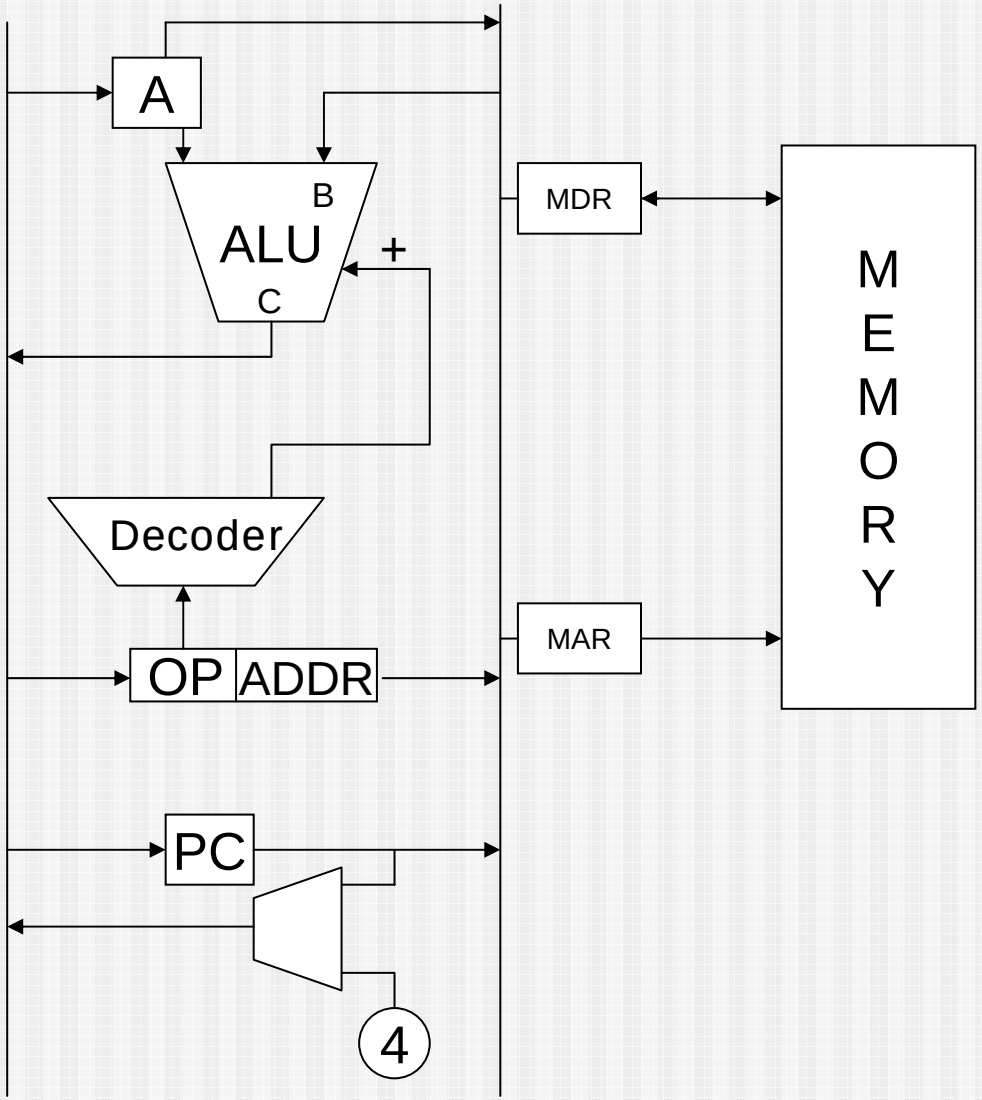
Two Bus Design(modify)



2 Bus Design

- With the added bus, 2 pairs of components may communicate in parallel.
- The additional bus does not change the current instruction set.
- Extra registers maybe need to allow data to be transferred from one bus to the other. In this example registers A, IP, and IR are used for this function.

2-BUS Architecture



Instruction Formats

- The Format of the current instructions are OP <ADDR>
- To provide instructions of the format

  ADD, R1, R2, R3

  and

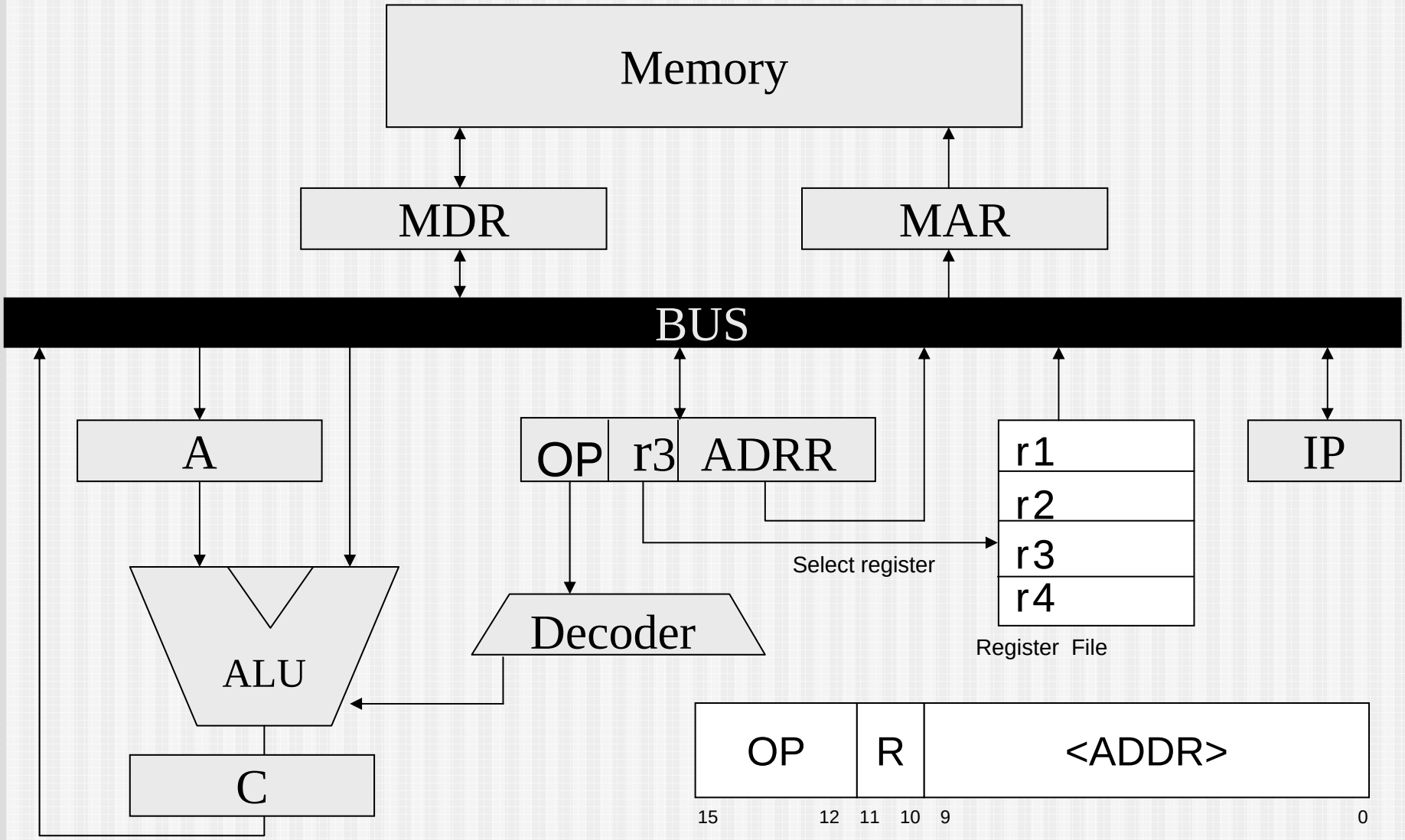
  STORE R3, <ADDR>

  where R1, R2 and R3 are registers we need to add an architecture to select the appropriate register

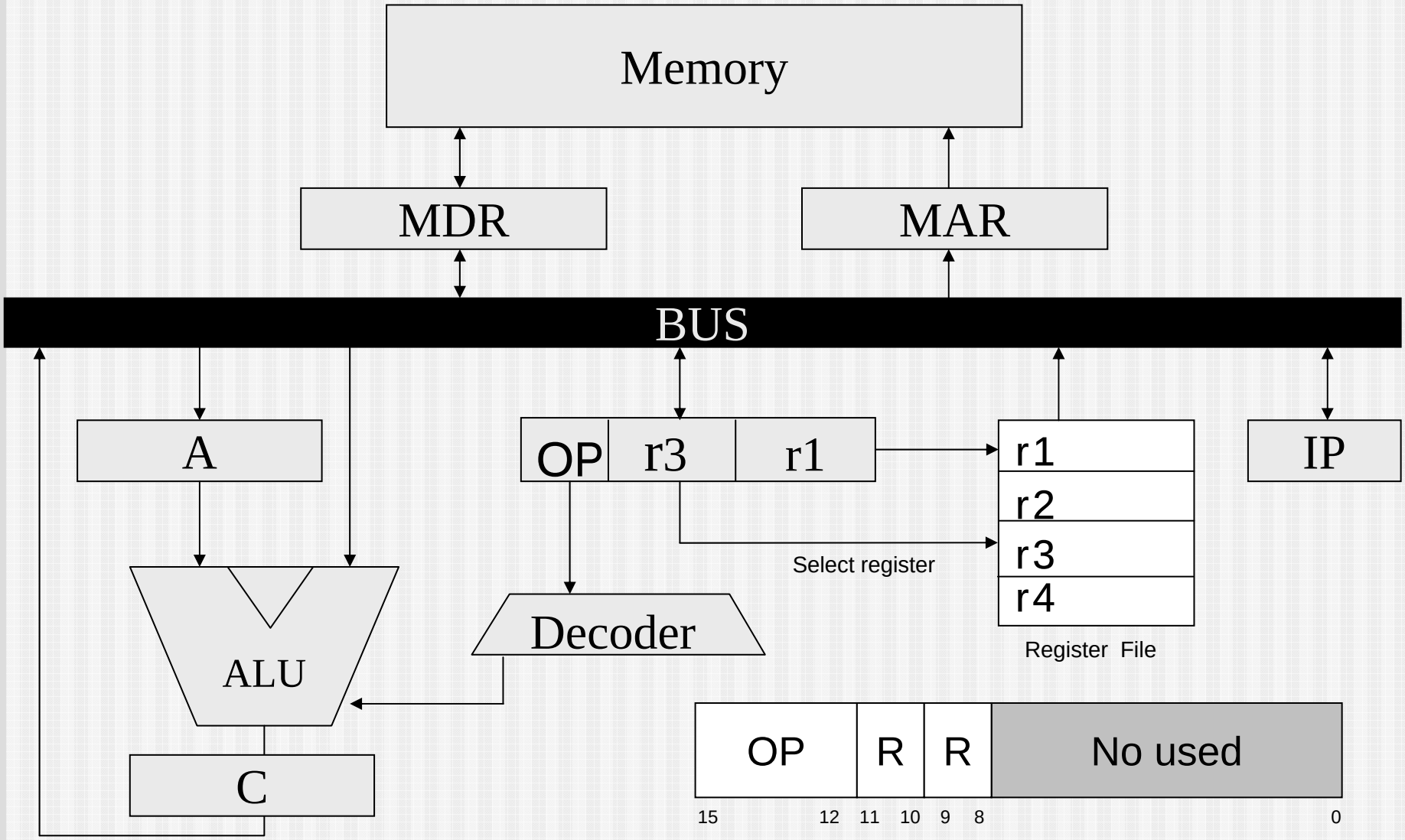
Register File

- By utilizing a register file you can organize the registers in one array
- The control unit allows us to select the appropriate register for the operation and load it into its target location (either memory or another register)
- This is a crucial aspect of the Load/Store architecture

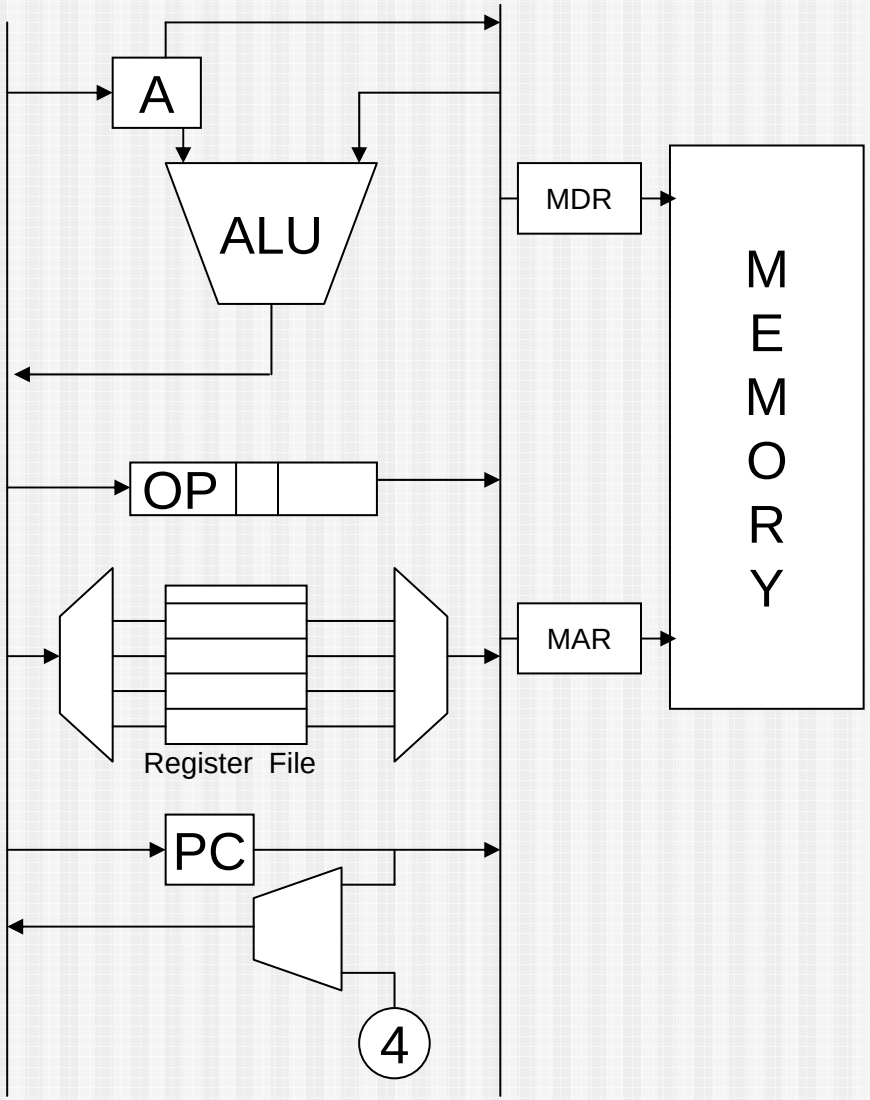
One Bus with Register File Load/Store Instruction Format



One Bus with Register File Register to Register Format

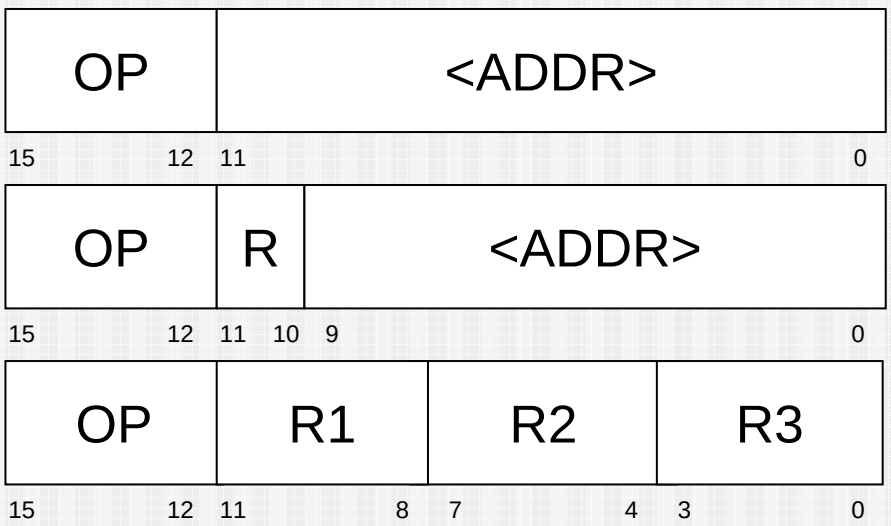


2-BUS Load/Store Architecture with Register File



Instruction Formats

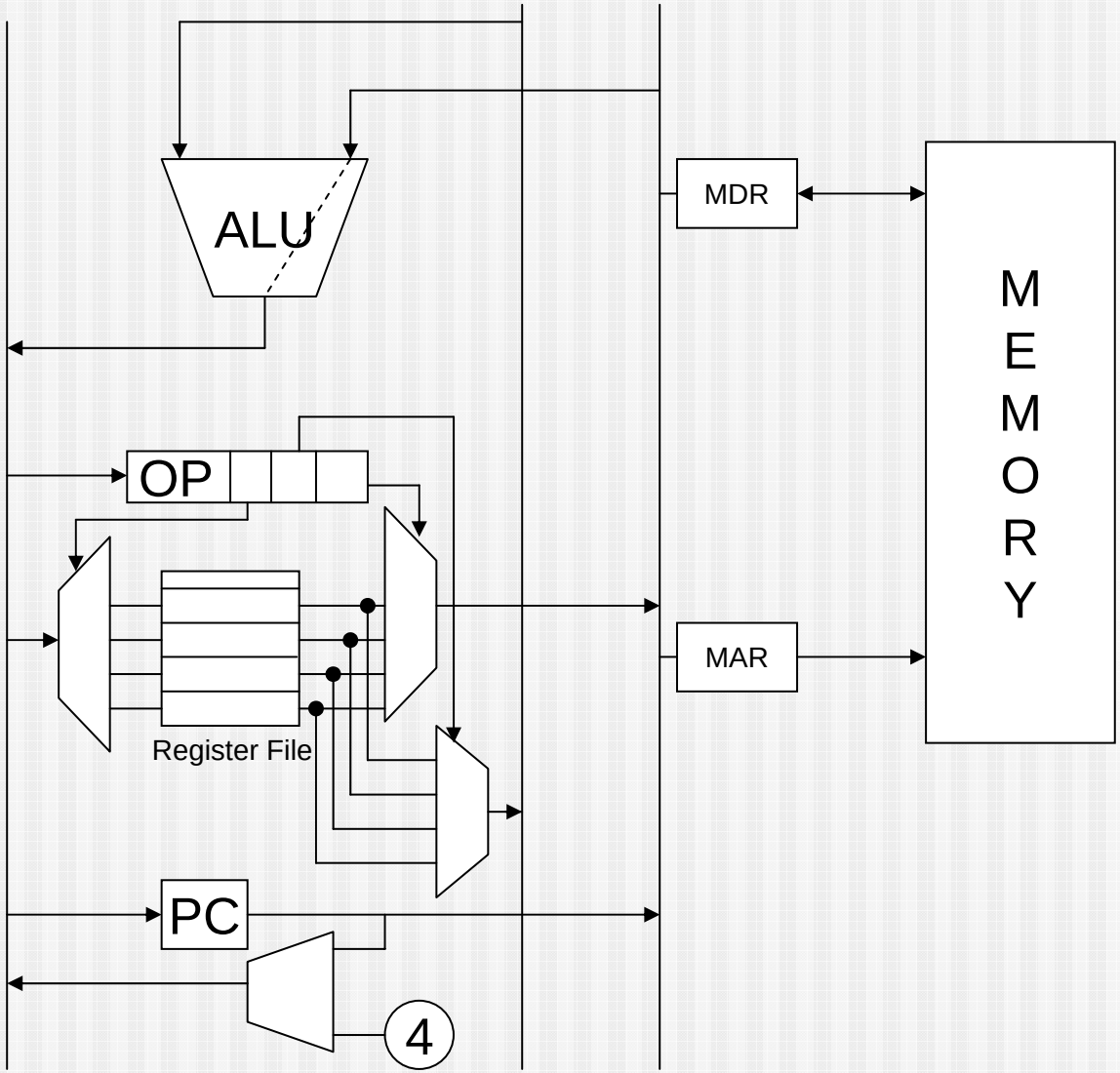
- The instruction format must be modified
- Three instruction formats are used



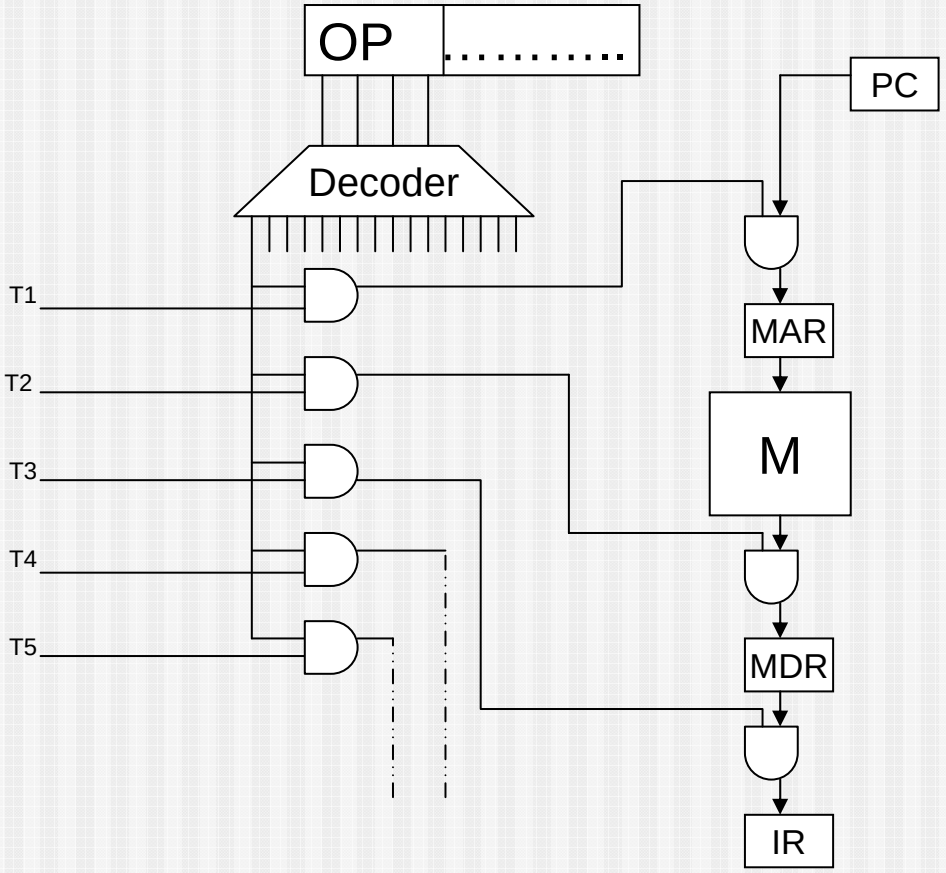
3-BUS Load/Store Architecture with Register File

- By adding a third BUS the system can load two registers at once using each BUS
- Two multiplexers need to be attached to the output of the register file: one connected to each BUS to send the contents of the target register to its destination
- Another multiplexer unit is added to send input from the third BUS into the correct location within the register file

3-BUS Load/Store Architecture with Register File

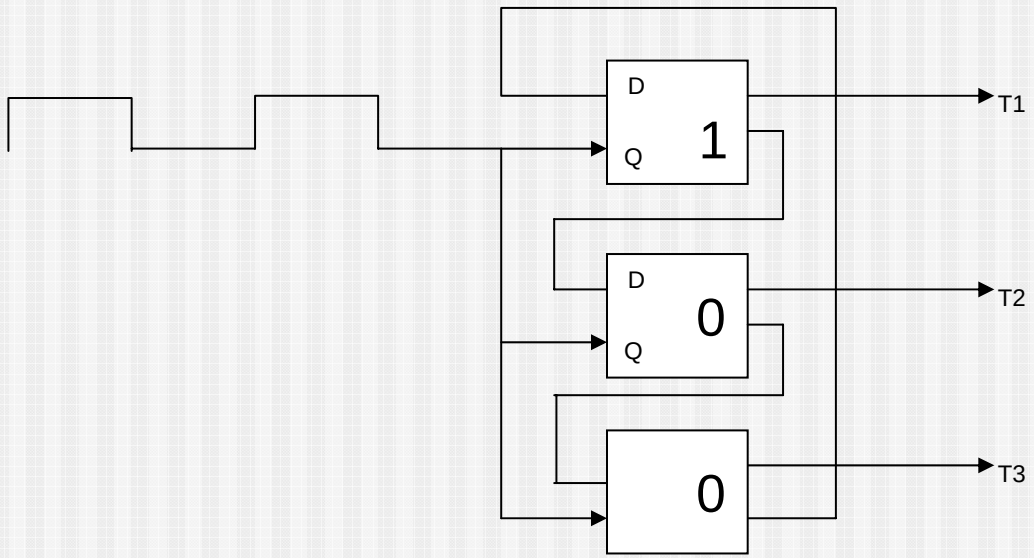


Control Unit



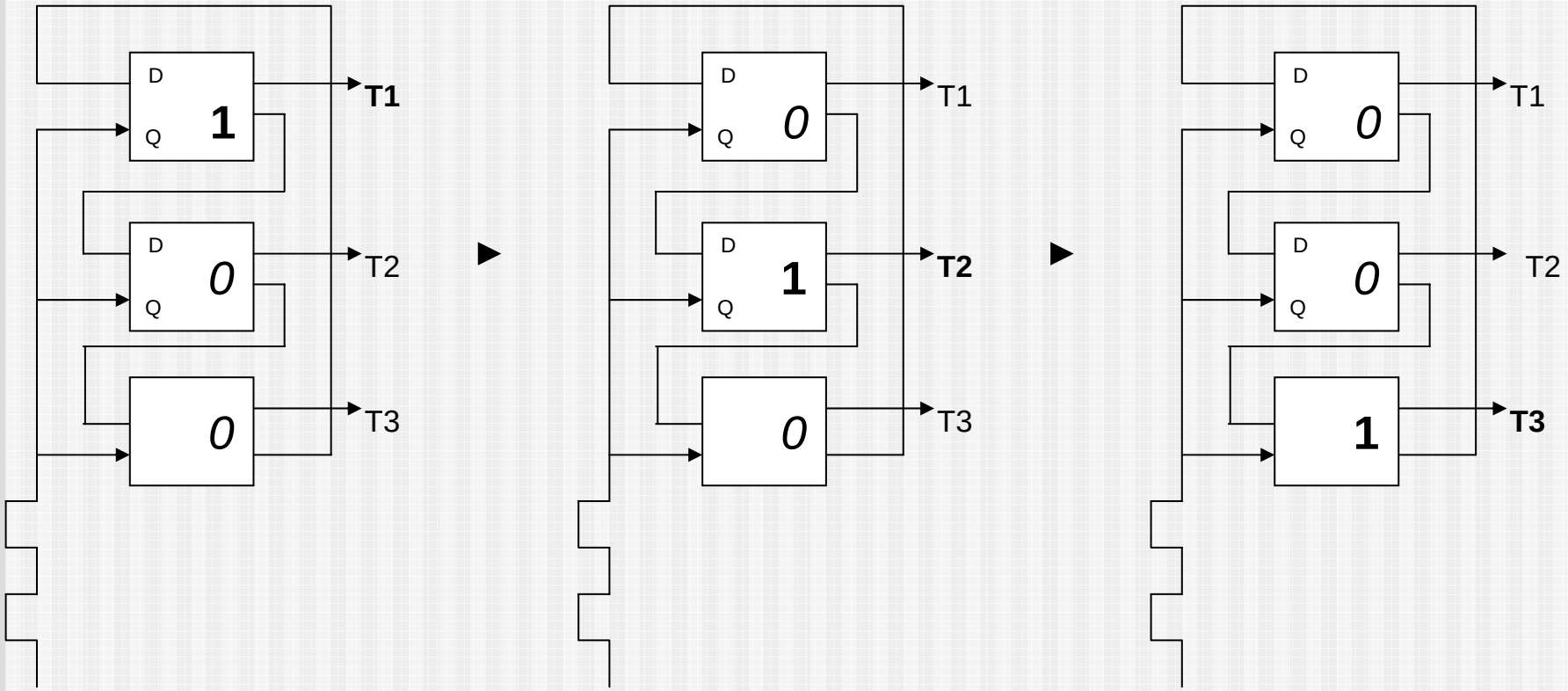
Clock Unit

- The Latch will only allow the change of the value on each clock tick
- By linking as a ring counter we create a clock distributor

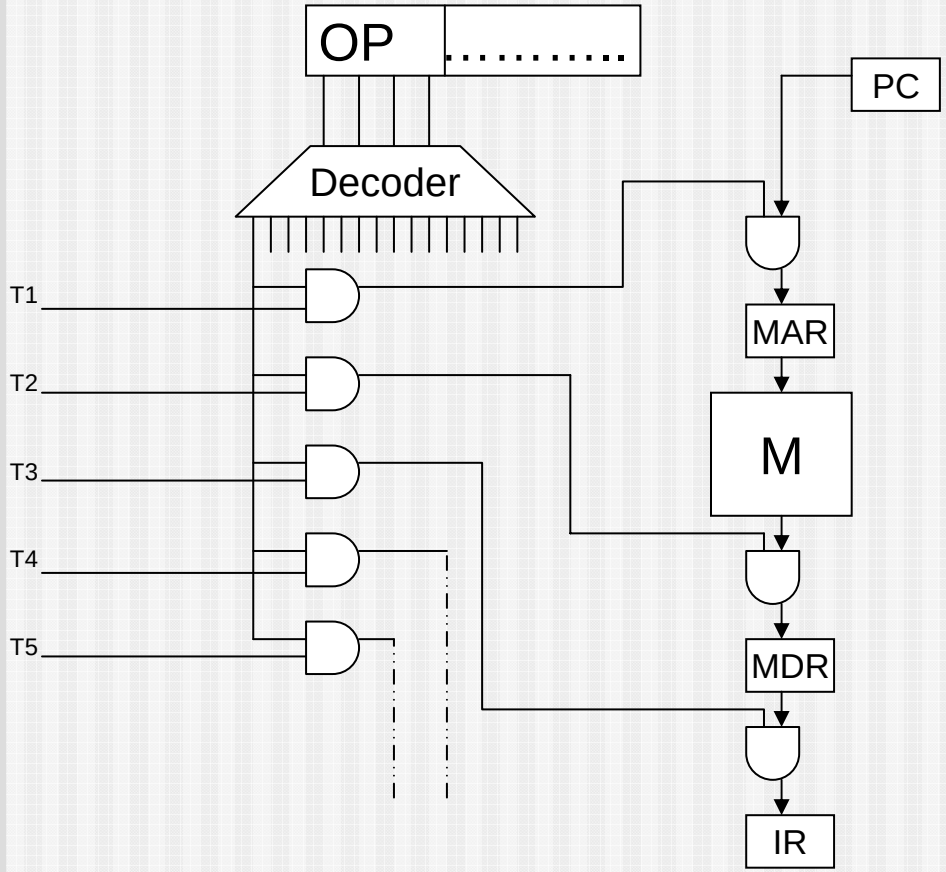


Clock Unit

- At every clock tick the active flip-flop changes, sending a signal to execute a different step of the current instruction in turn

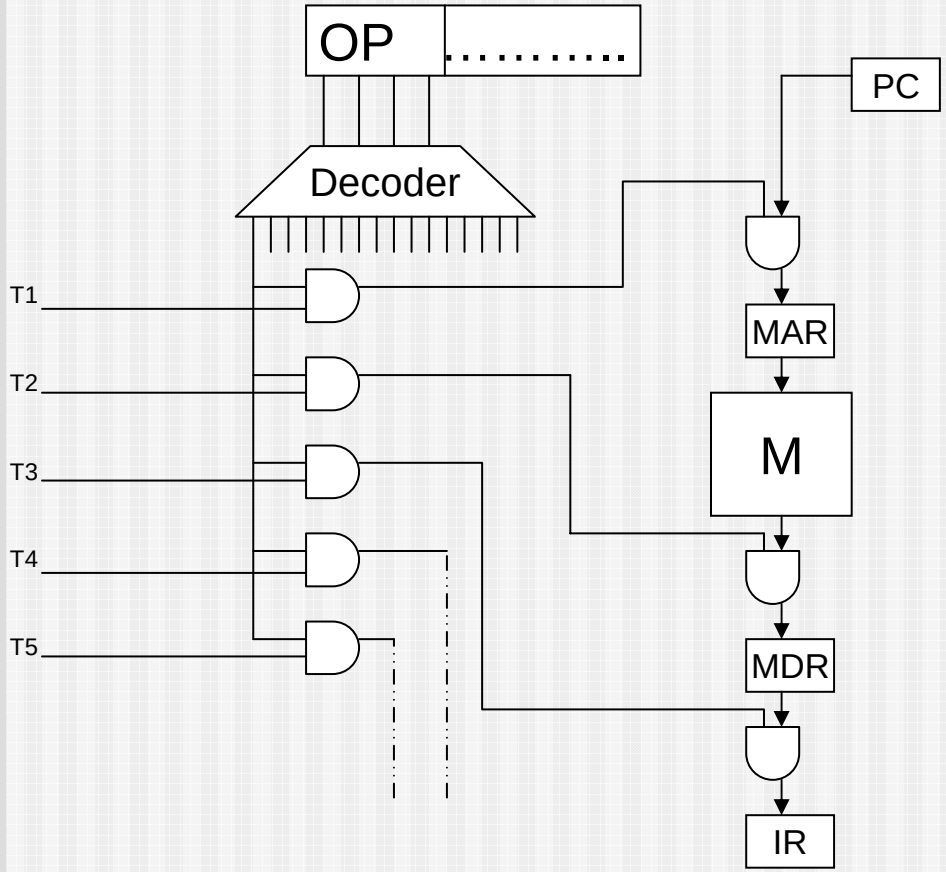


Control Unit



Each line emerging from the decoder represents an operation. When the decoder is set to that operation, it sends voltage down the appropriate channel which is ANDed with the signals coming from the clock. This allows for precise timing in the step executions of instructions and makes synchronization among components possible.

Control Unit



Fetch Cycle:

T1: MAR <- PC

T2: MDR <- M[MAR]

T3: IR <- MDR

T4: MAR <- IR.ADDR

T5: Decoder <- IR.OP

The chain of flip-flops will be as long in clock ticks as the longest command takes to execute.

One-Address Format

- The operation is 4 bits and the remaining 12 are address bits
- This gives the availability of 2^4 operations and 2^{12} memory locations

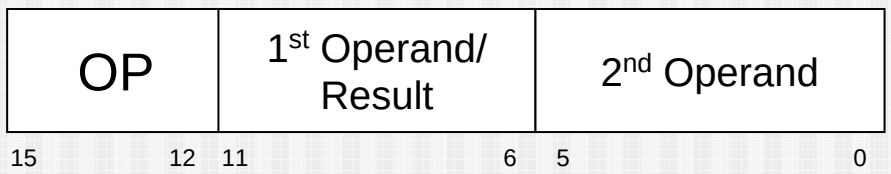
One-Address Format

15-12	Command
0000	*Fetch*
0001	Add
0010	Sub
0011	Mult
…	
1111	

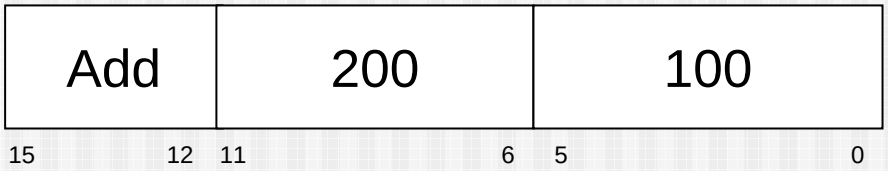
- 0000 is the Fetch operation(hidden instruction)
- It is a hidden instruction that cannot be accessed by the user

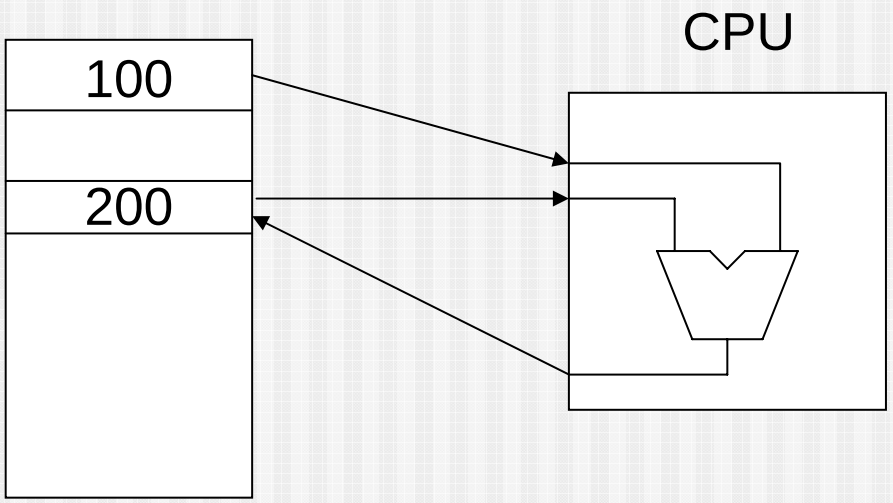
Two-Address Format

- The operation is performed on the memory addresses of the first and second operands and the value is stored back in the location specified by the first operand field



Two-Address Format





This is analogous to:

Load 200

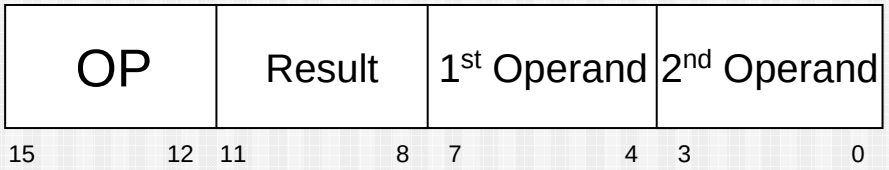
Add 100

Store 200

all performed in one command

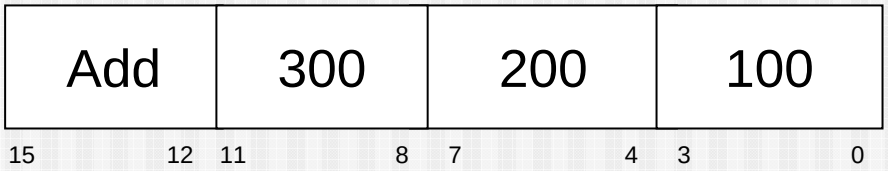
Three-Address Format

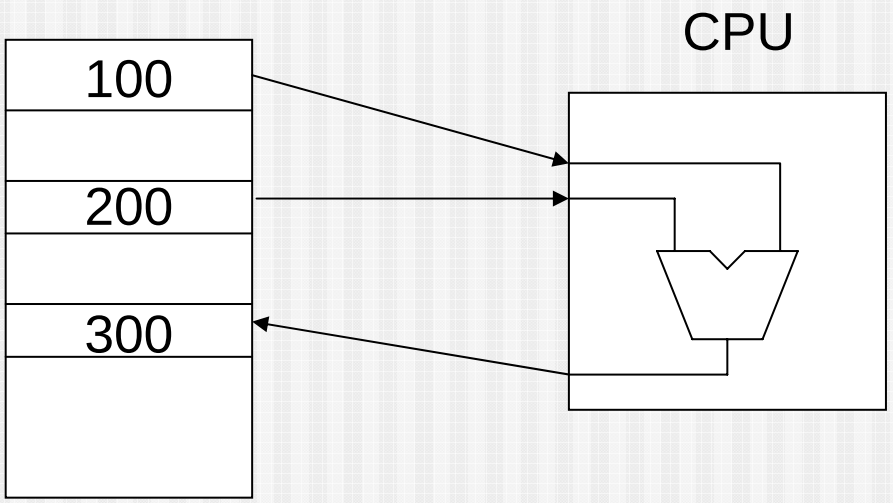
- The operation is performed on the memory addresses of the first and second operands and the value is stored in the location specified by the result address field



- This format is not often used because it requires three memory accesses per operation (which is very slow)

Three-Address Format





This is analogous to:

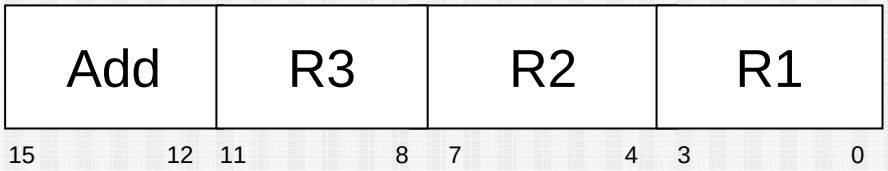
Load 200

Add 100

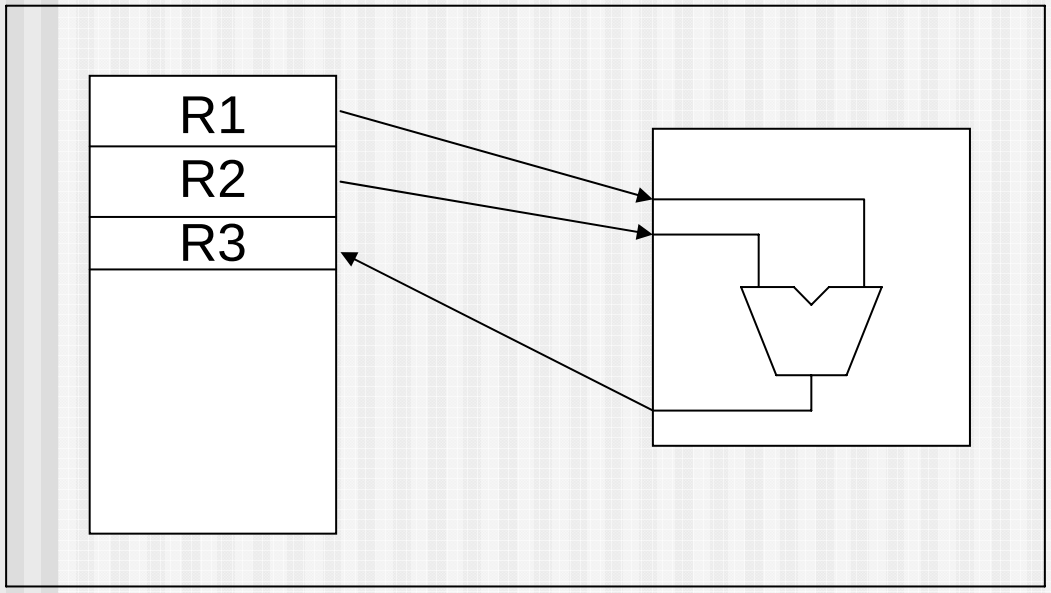
Store 300

all performed in one command

Another Three-Address Format



CPU



In a Load/Store machine, the addresses are often locations in the register file rather than in memory. This is very fast on a machine with multiple-BUS architecture.

Addressing Modes

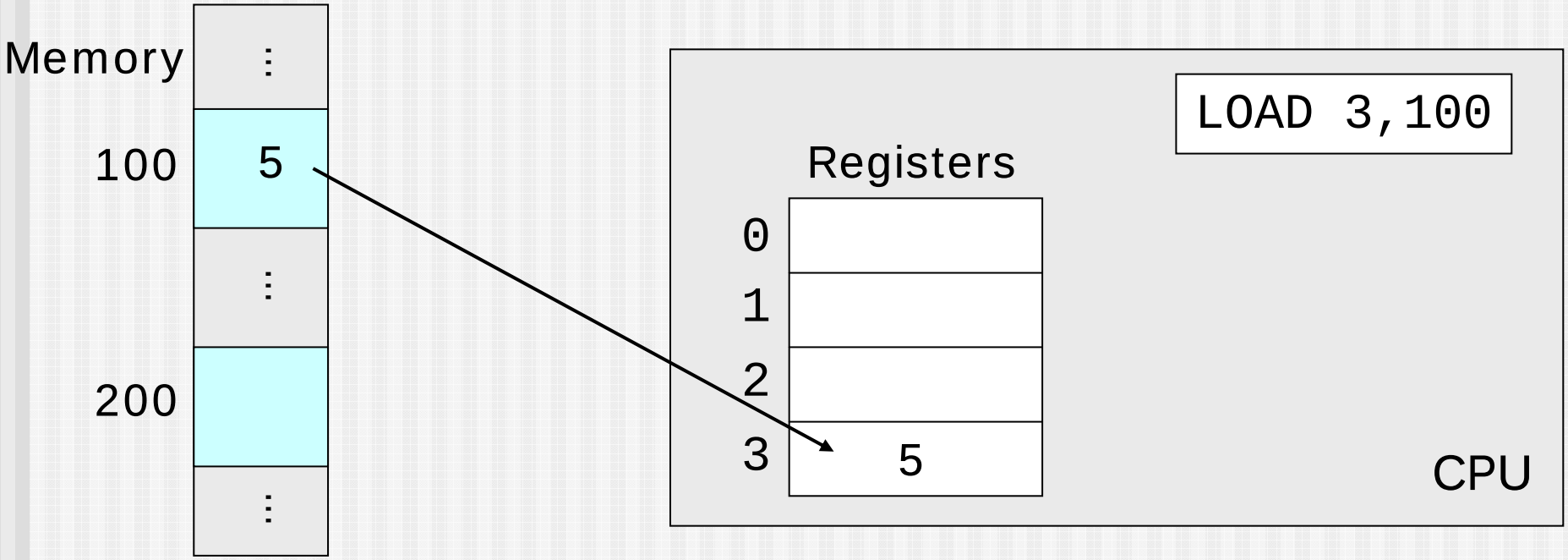
- Direct
- Immediate
- Indirect
- Register Direct
- Register Indirect
- Register Indirect plus Offset

Direct Addressing

Used for manipulating an absolute address.

Example: `LOAD R1, <ADDR>`

will load the contents of <ADDR> into R1

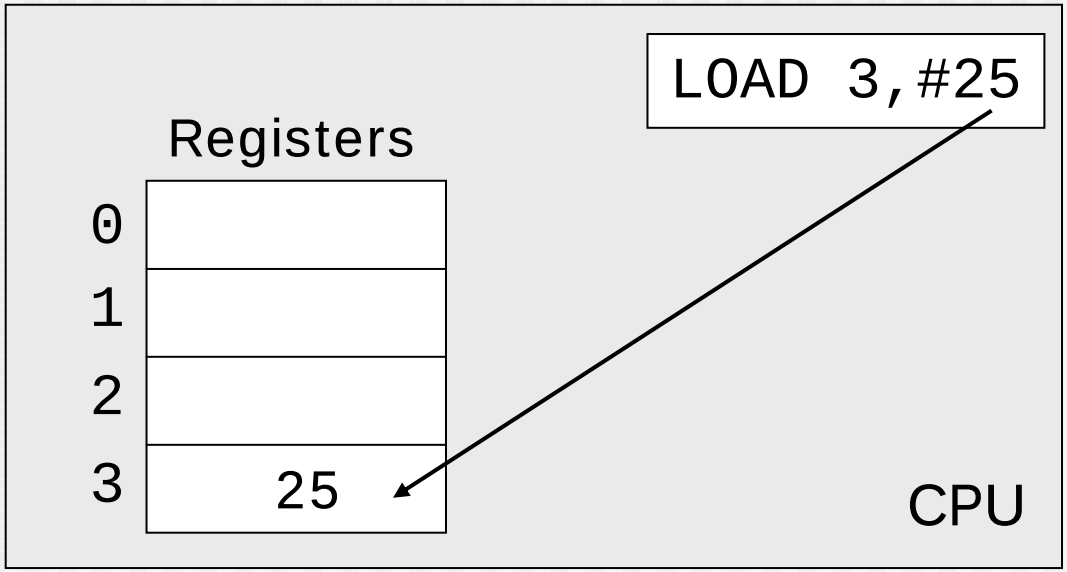


Immediate Addressing

Used when dealing with constants.

Example: `LOAD R1, #value`

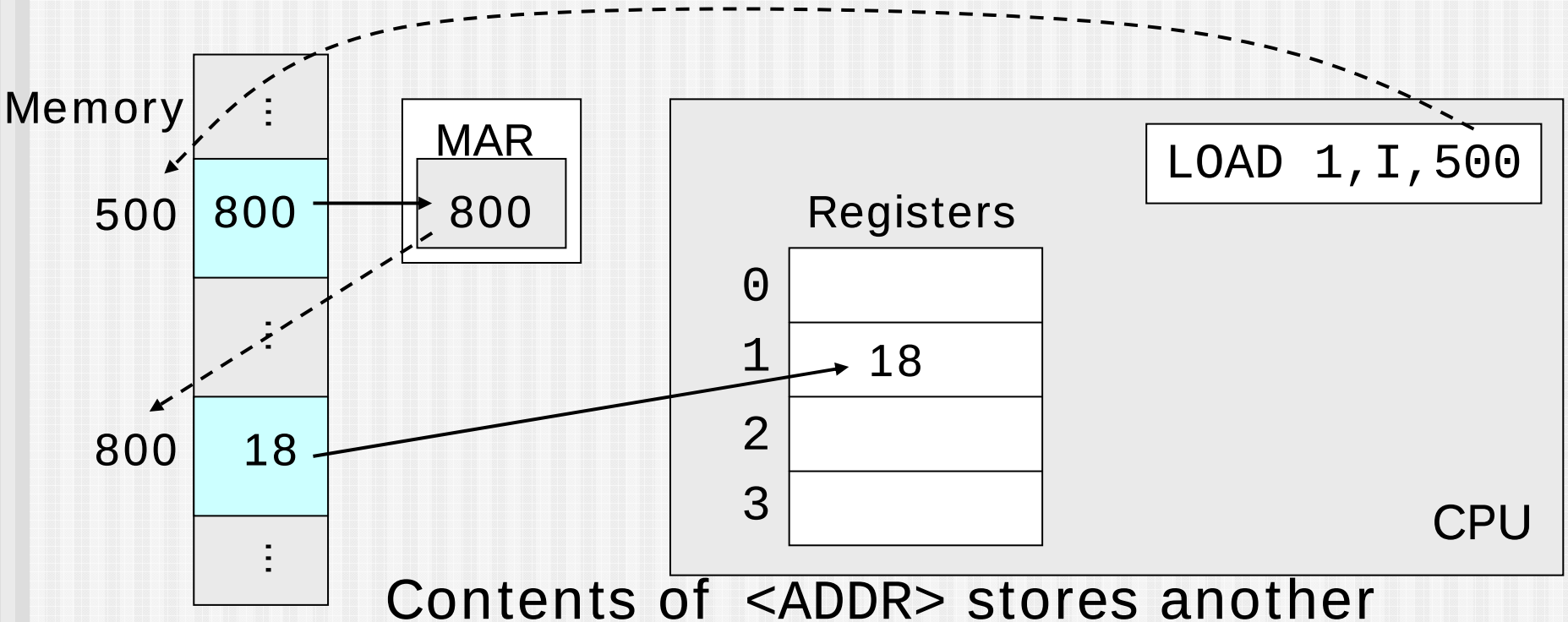
will load `value` into `R1`.



Indirect Addressing

Typically used for accessing a value through a pointer.

Example: LOAD R1,I,<ADDR>



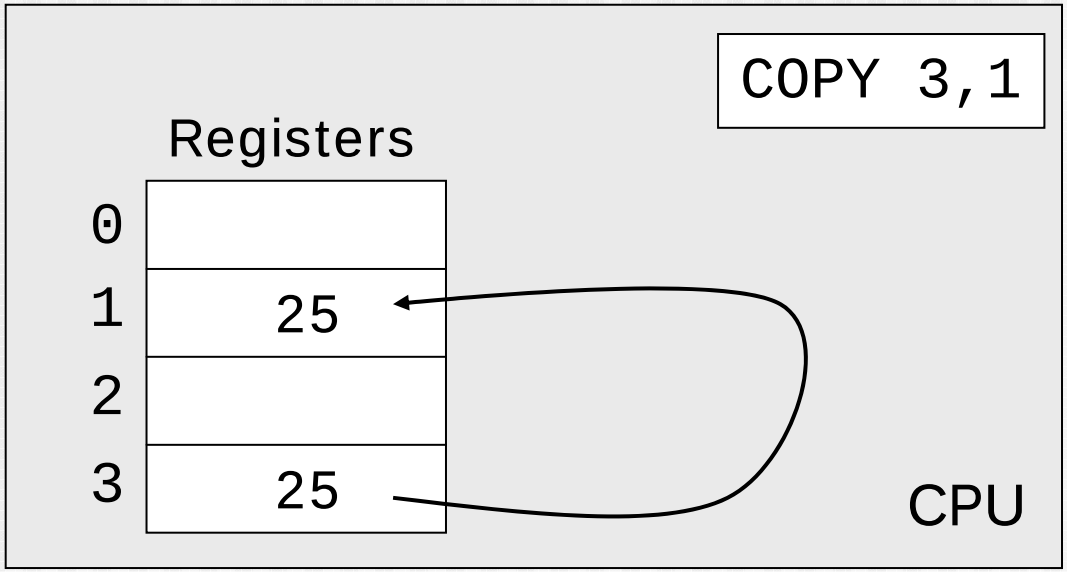
Contents of <ADDR> stores another address whose contents will be loaded into R1

Register Direct Addressing

Used to copy the contents of one register into another.

Example: `COPY R2,R1` or `MOVE R2,R1`

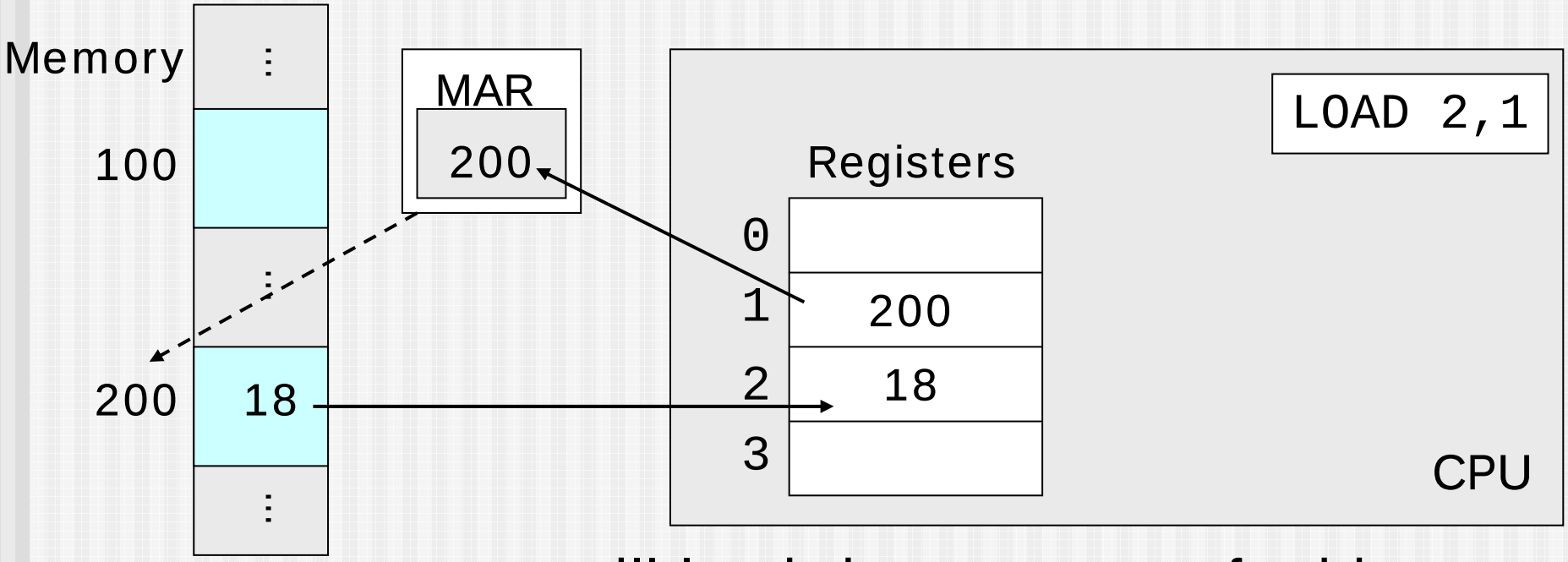
will copy contents of `R1` into `R2`.



Register Indirect Addressing

Typically used for accessing a list of consecutive memory locations.

Example: LOAD R2,<R1>

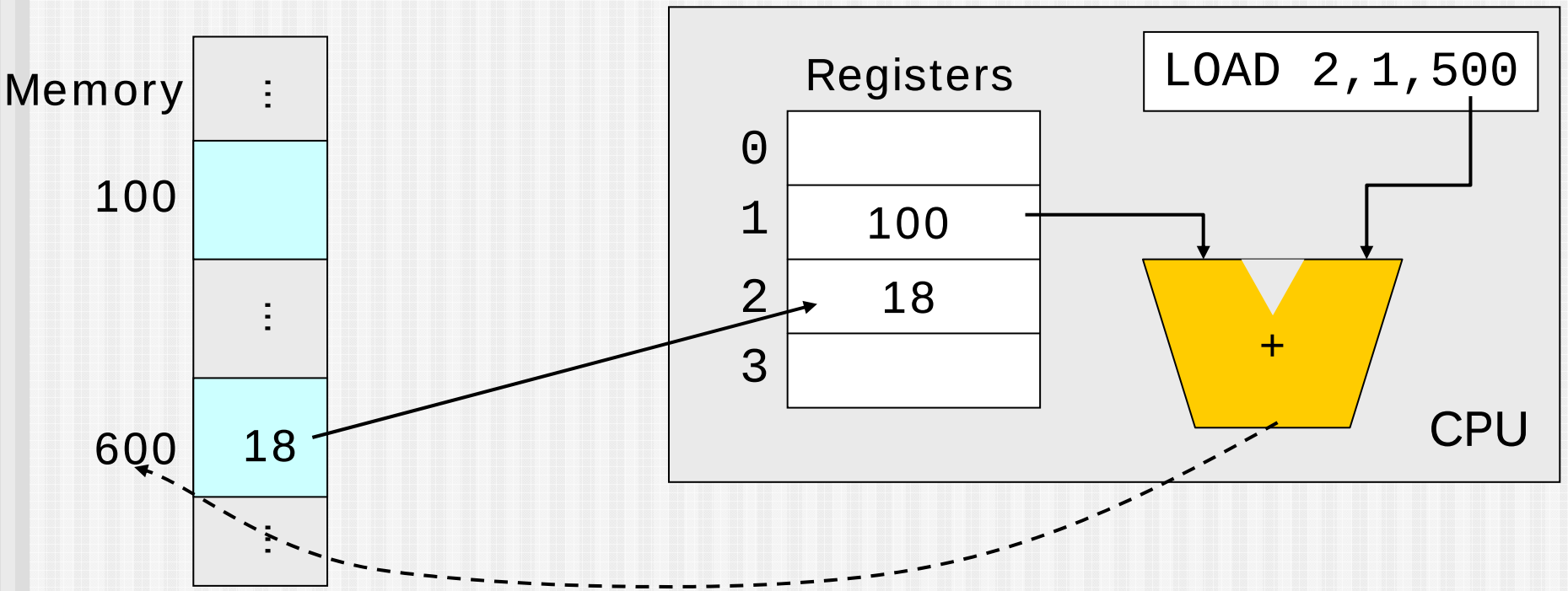


will load the contents of address stored in R1 into R2

Register Indirect Addressing plus Offset

Typically used when accessing array and structures.

Example: `LOAD R2,R1,offset`



will load the contents of address stored in R1+`offset` into R2