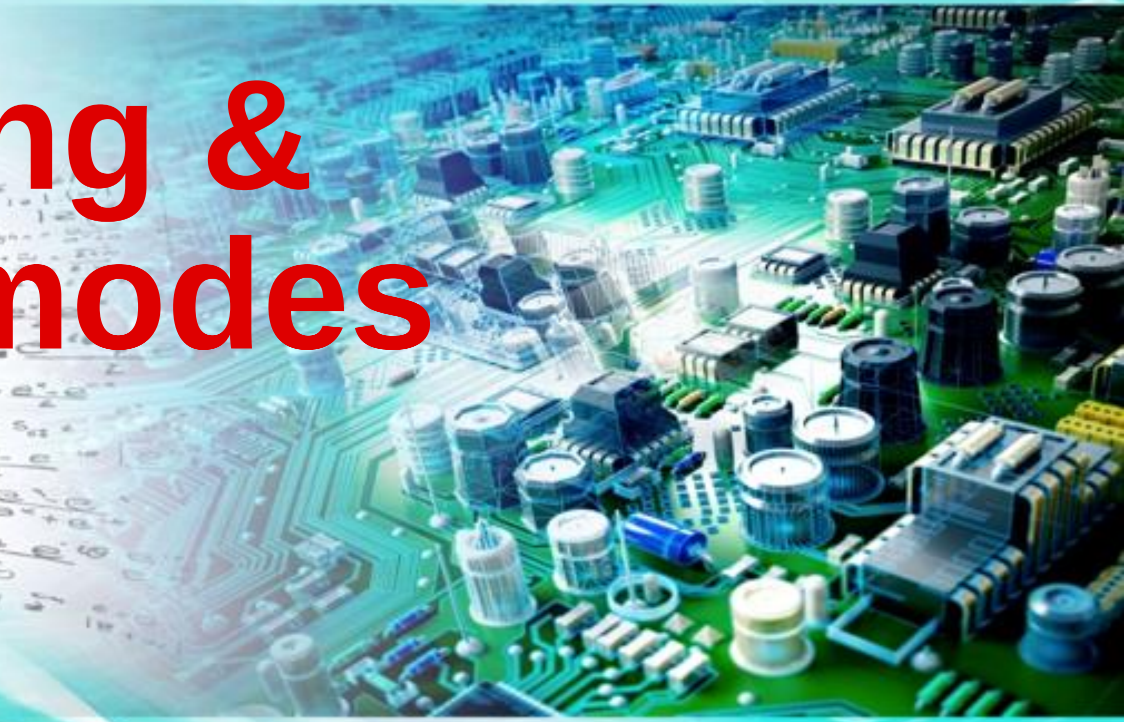
Memory addressing & CPU addressing modes

TI Precision Labs – Microcontrollers

Presented by Brandon Fisher

Prepared by Evan Lew

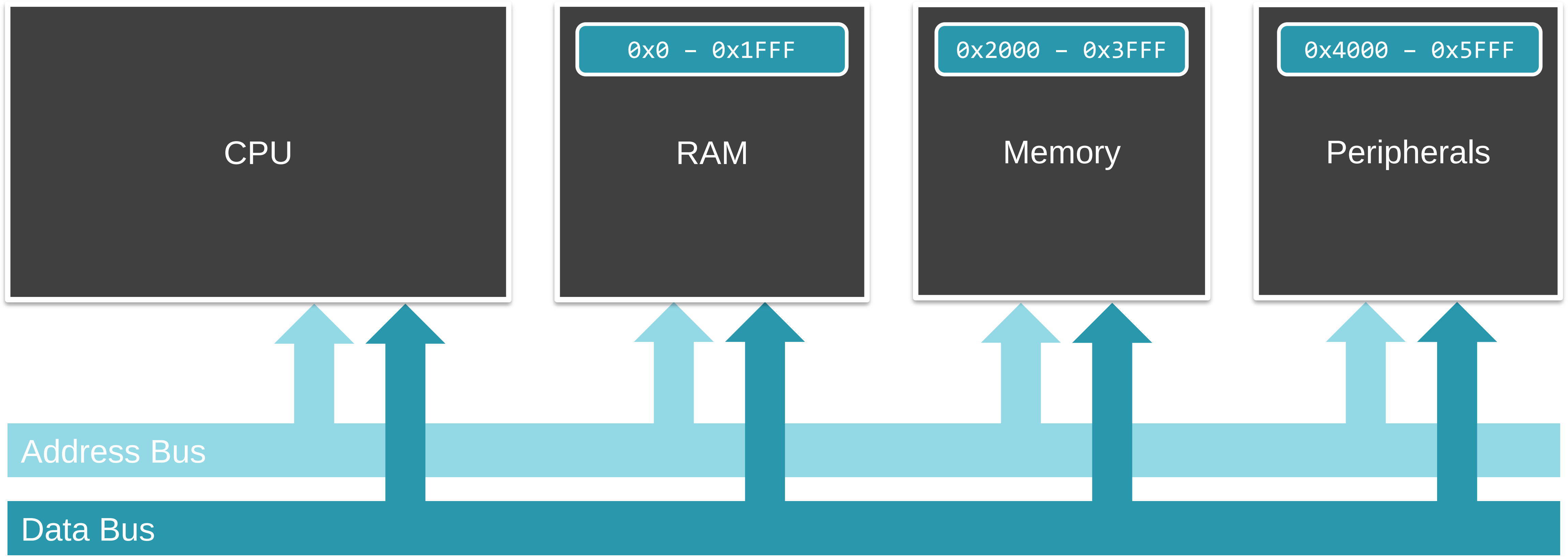


Memory addressing review

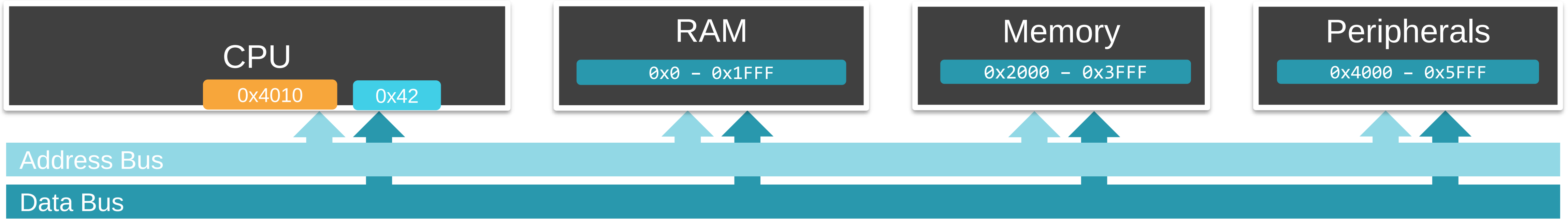
N Memory Ln
1000
1002
1004
1006

Texas Instruments

Memory addressing review



Examples of addressing



Peripheral configuration

Example:

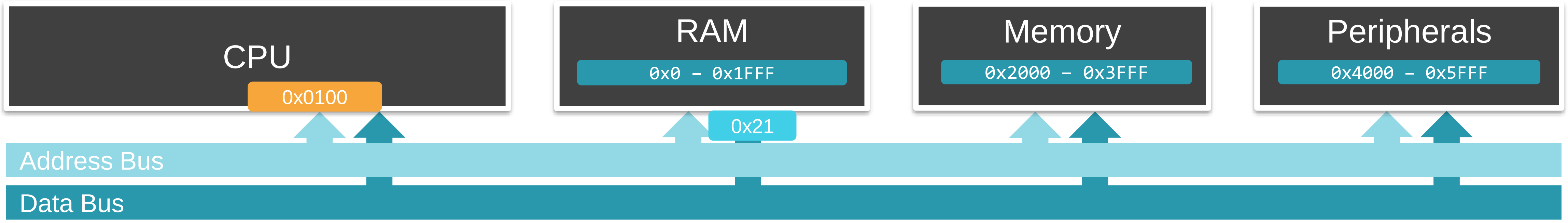
`I2C_CTRL = 0x42;`

Addressing:

The address of `I2C_CTRL` is `0x4010`

`I2C_CTRL` represents the address of a specific register where the value `0x42` will be written

Examples of addressing



Peripheral configuration

Example:

```
I2C_CTRL = 0x42;
```

Addressing:
The address of `I2C_CTRL` is `0x4010`

`I2C_CTRL` represents the address of a specific register where the value `0x42` will be written

Global variable access

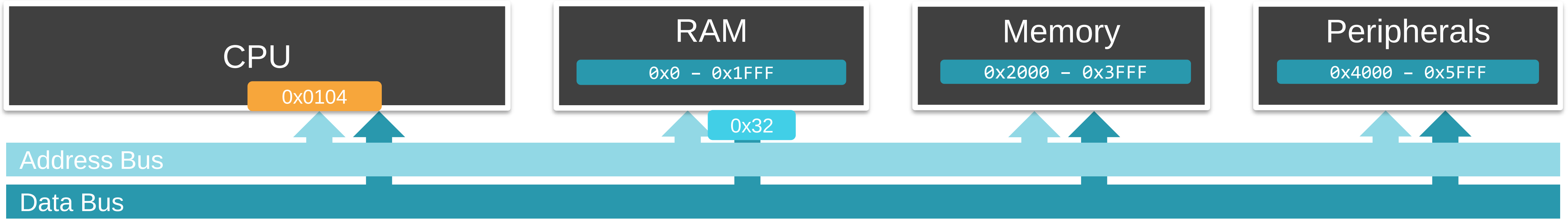
Example:

```
if (global_cnt == 0x21) {
  do_something();
}
```

Addressing:
The address of `global_cnt` is `0x0100`.

The CPU must first read global_cnt from RAM to check its value. In this case let's assume the value is `0x21`.

Examples of addressing



Peripheral configuration

Example:

```
I2C_CTRL = 0x42;
```

Addressing:
The address of `I2C_CTRL` is `0x4010`

`I2C_CTRL` represents the address of a specific register where the value `0x42` will be written

Global variable access

Example:

```
if (global_cnt == 0x21) {
  do_something();
}
```

Addressing:
The address of `global_cnt` is `0x0100`.

The CPU must first read global_cnt from RAM to check its value. In this case let's assume the value is `0x21`.

C pointer

Example:

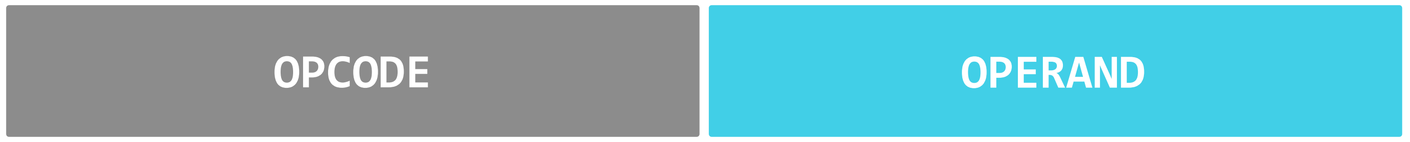
```
int is_positive(int *val) {
  return *val > 0;
}
```

Addressing:
The address of `val` is `0x0104`.

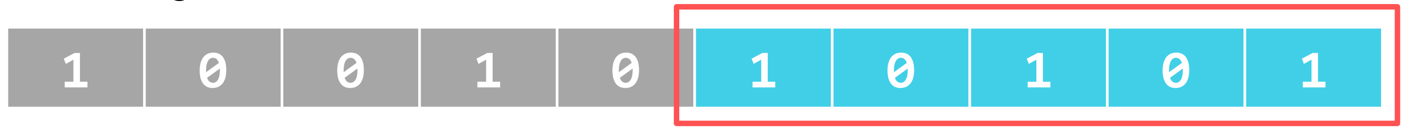
The CPU fetches the data before it performs the compare operation. Let's say that data is `0x32`

CPU addressing modes

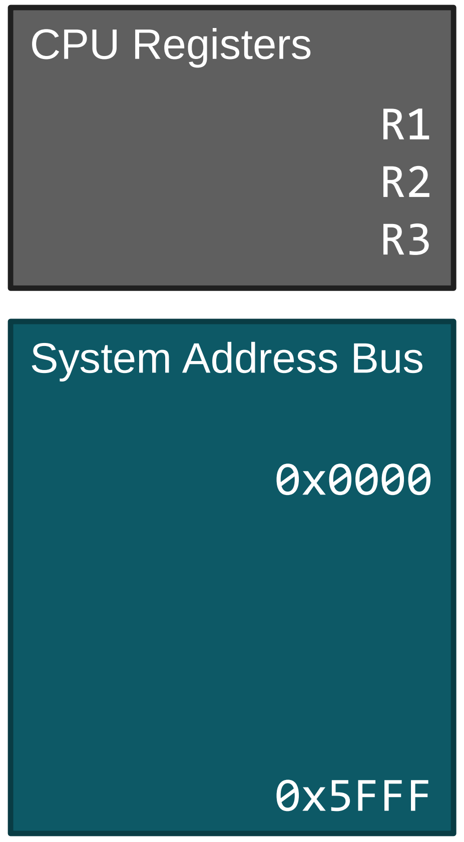
Simplified view of an instruction:



Encoding of an instruction:



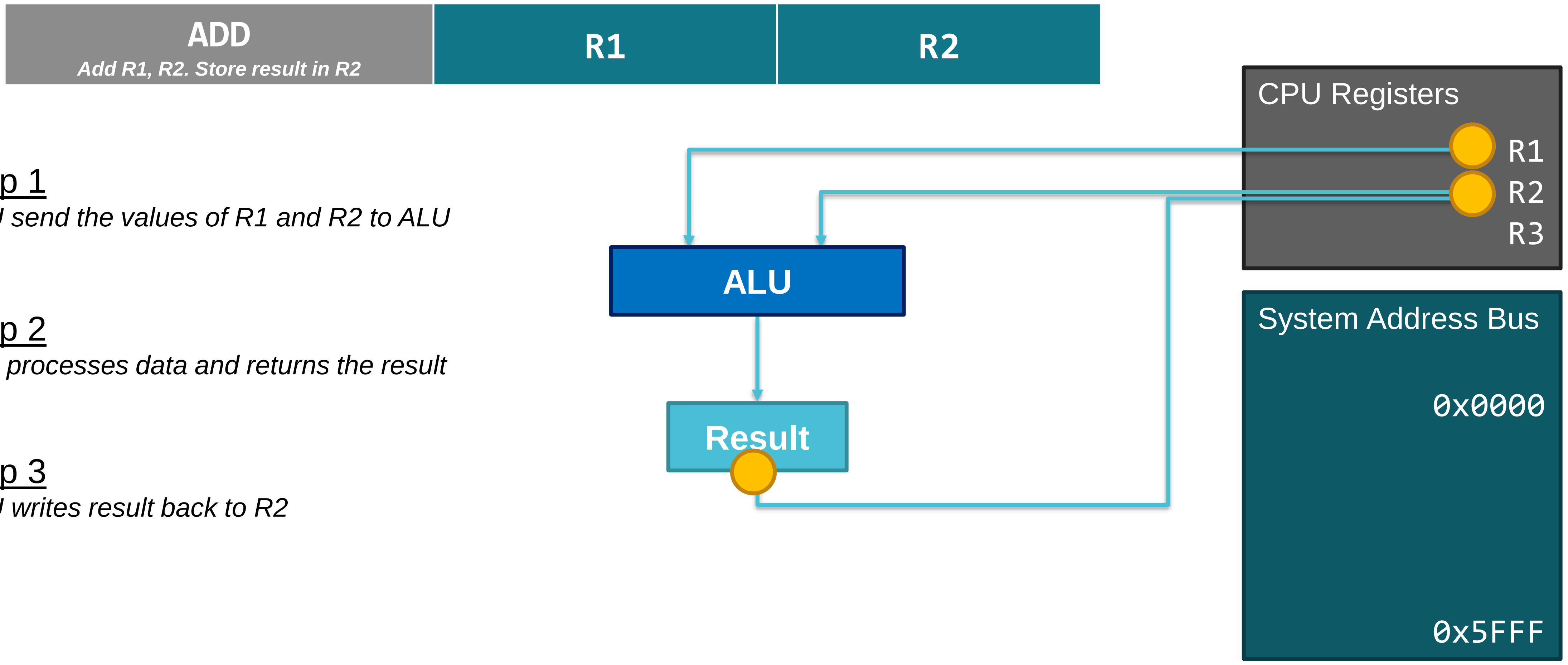
- Determines which instruction the CPU will run
- Determines what data the CPU will process



Key concept:
Addressing mode are the ways a CPU can access data in the system

Register addressing mode

→ Register contents are operand



<u>Step 1</u>
CPU send the values of R1 and R2 to ALU

<u>Step 2</u>
ALU processes data and returns the result

<u>Step 3</u>
CPU writes result back to R2

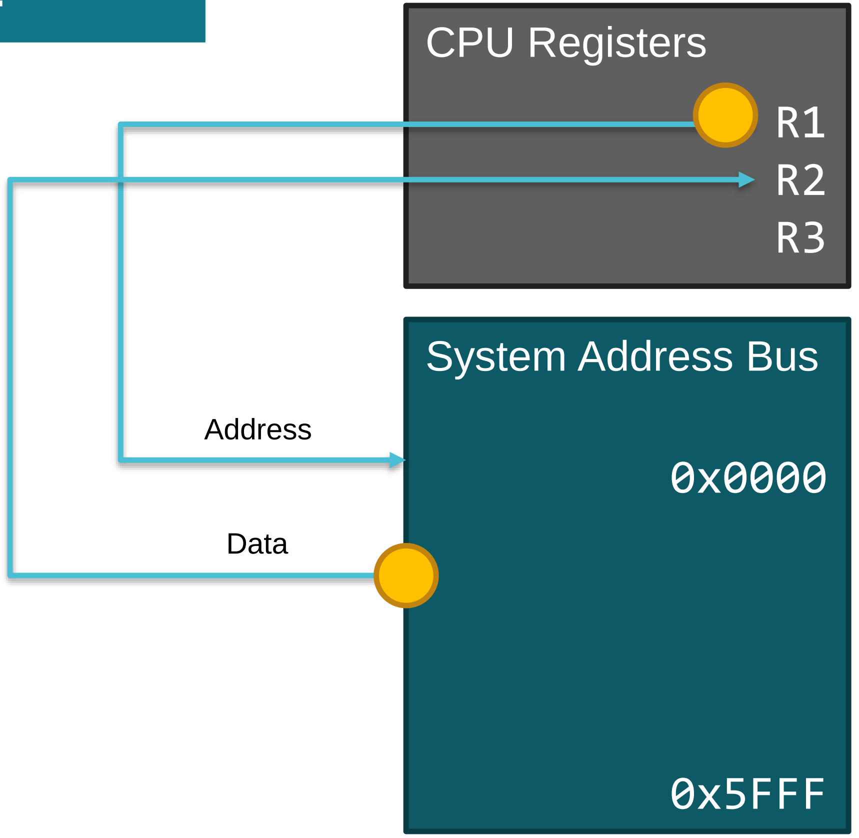
Indirect register addressing mode

→ Register contents is an address pointer to the operand

MOV *Load data from the address in R1 into R2*	@R1	R2

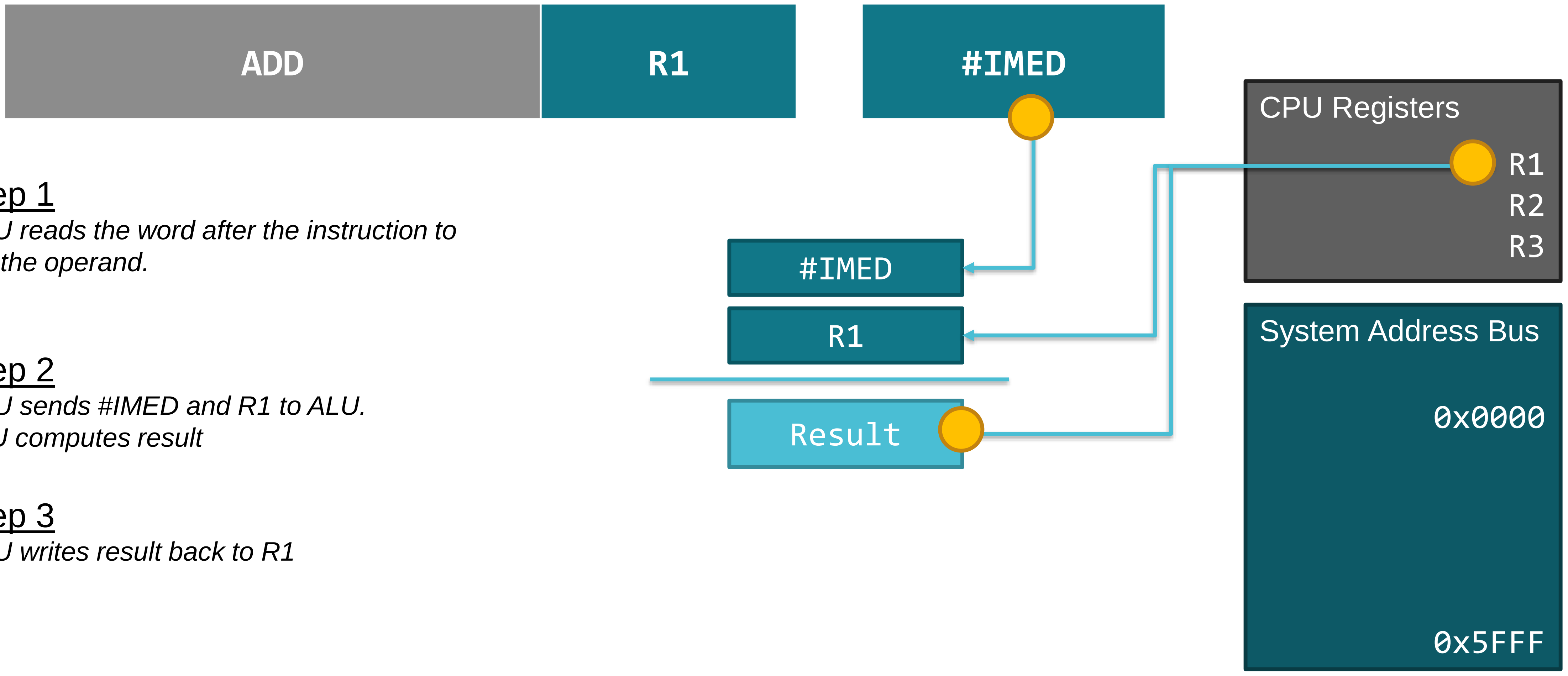
Step 1
CPU requests data at the address in R1

Step 2
CPU writes data into R2



Immediate addressing mode

→ Data following instruction is the operand



<u>Step 1</u>

CPU reads the word after the instruction to get the operand.

<u>Step 2</u>

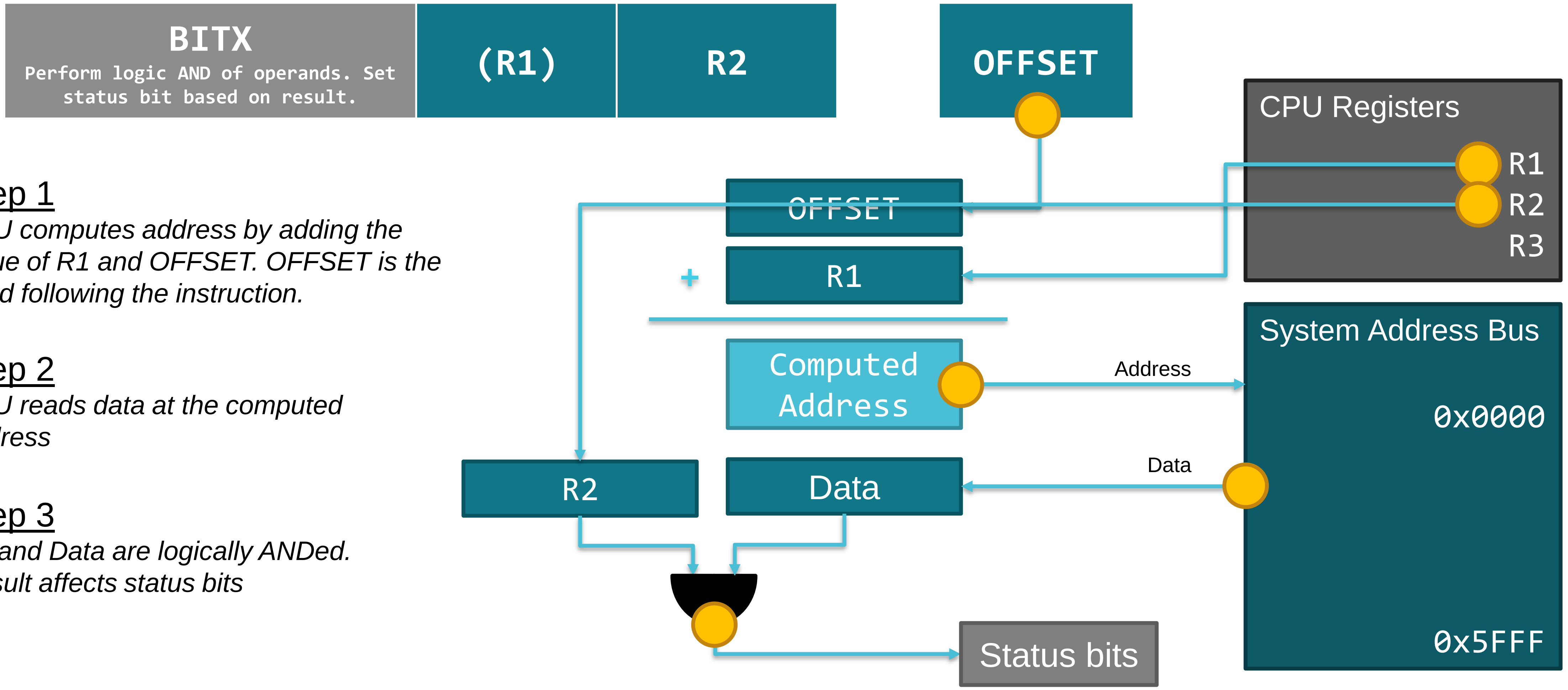
CPU sends #IMED and R1 to ALU.
ALU computes result

<u>Step 3</u>

CPU writes result back to R1

Indexed addressing mode

→ Data at address in the register plus an offset



__Step 1__
CPU computes address by adding the value of R1 and OFFSET. OFFSET is the word following the instruction.

__Step 2__
CPU reads data at the computed address

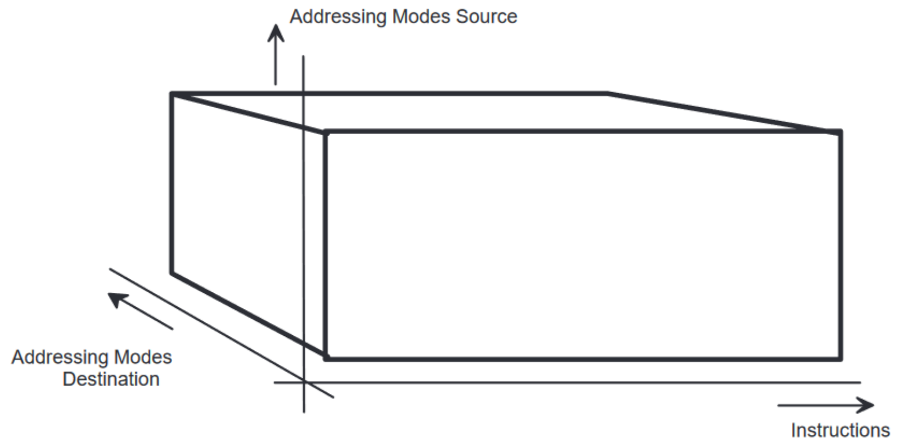
__Step 3__
R2 and Data are logically ANDed. Result affects status bits

Addressing mode description

Addressing Mode	Syntax	Description
Register Mode	Rn	Register contents are operand
Indexed Mode	X(Rn)	(Rn + X) points to the operand. X is stored in the next word
Symbolic Mode	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed Mode X(PC) is used
Absolute Mode	&ADDR	The word following the instruction contains the absolute address
Indirect Register	@Rn	Rn is used as a pointer to the operand
Indirect Autoincrement	@Rn+	Rn is used as a pointer to the operand. Rn is incremented afterwards
Immediate Mode	#N	The word following the instruction contains the immediate constant N. Indirect Autoincrement Mode @PC+ used

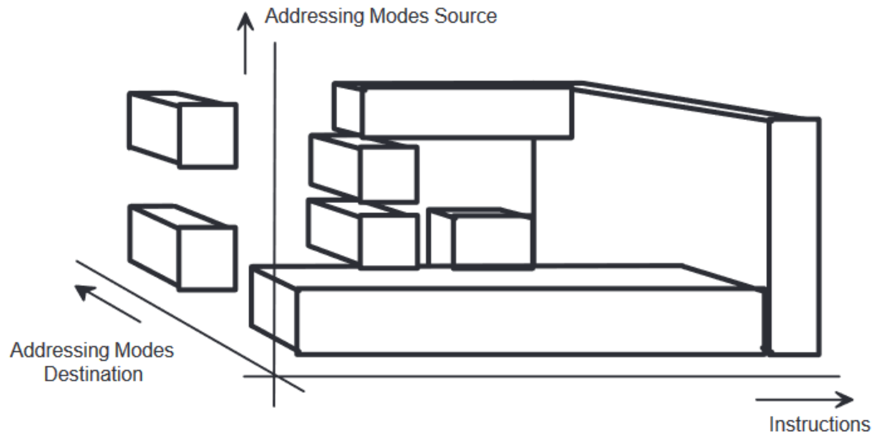
Orthogonal architecture:

Instructions implement all addressing modes for all operands



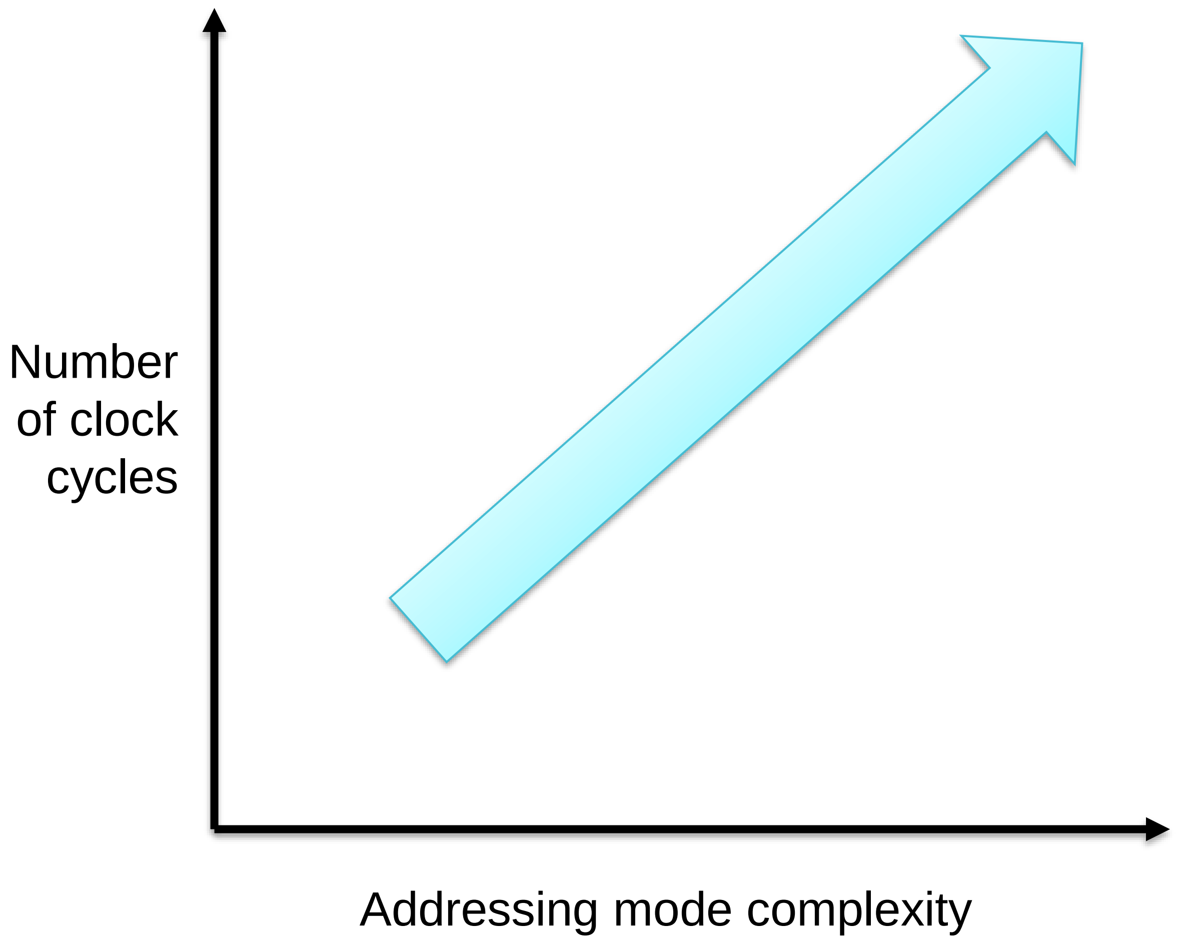
Non-Orthogonal architecture:

Instructions implement a subset of addressing modes



Addressing mode speed comparison

Addressing Mode		Clock Cycles
Source	**Destination**	
Register	Register	1
	Program counter	3
	Indexed	4
	Absolute	4
Indirect register	Register	2
	Program counter	4
	Indexed	5
	Absolute	5
Indirect register with autoincrement	Register	2
	Program counter	4
	Indexed	5
	Absolute	5
Immediate	Register	2
	Program counter	3
	Indexed	5
	Absolute	5



Addressing modes in practice

C Code:

```
int is_odd(int *val) {
    if (*val & 0x1) {
        return 1;
    } else
        return 0;
}
```

Assembly:

```
DECD.W   SP
MOV.W    R12,0x0000(SP)
MOV.W    #1,R15
BIT.W    @R12,R15
JEQ      ($C$L1)
MOV.W    #1,R12
JMP      ($C$L2)
CLR.W    R12
INCD.W   SP
RETA
```

Description
Load the value ‘1’ into R15
Source operand addressing mode
Immediate
Destination addressing mode
Register

Texas Instruments

Addressing modes in practice

C Code:

```
int is_odd(int *val) {
    if (*val & 0x1) {
        return 1;
    } else
        return 0;
}
```

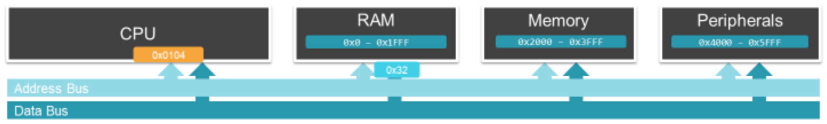
Assembly:

```
DECD.W   SP
MOV.W    R12,0x0000(SP)
MOV.W    #1,R15
BIT.W    @R12,R15
JEQ      ($C$L1)
MOV.W    #1,R12
JMP      ($C$L2)
CLR.W    R12
INCD.W   SP
RETA
```

Description
Test if bits are set in both @R12 and R15
Source operand addressing mode
Register indirect
Destination addressing mode
Register

Addressing mode review

Examples of addressing



Peripheral configuration

Example:
`I2C_CTRL = 0x42;`

Addressing:
The address of `I2C_CTRL` is `0x4010`

`I2C_CTRL` represents the address of a specific register where the value `0x42` will be written

Global variable access

Example:
```
if (global_cnt == 0x21) {
  do_something();
}
```

Addressing:
The address of `global_cnt` is `0x0100`.

The CPU must first read global_cnt from RAM to check its value. In this case let's assume the value is `0x21`.

C pointer

Example:
```
int is_positive(int *val) {
  return *val > 0;
}
```

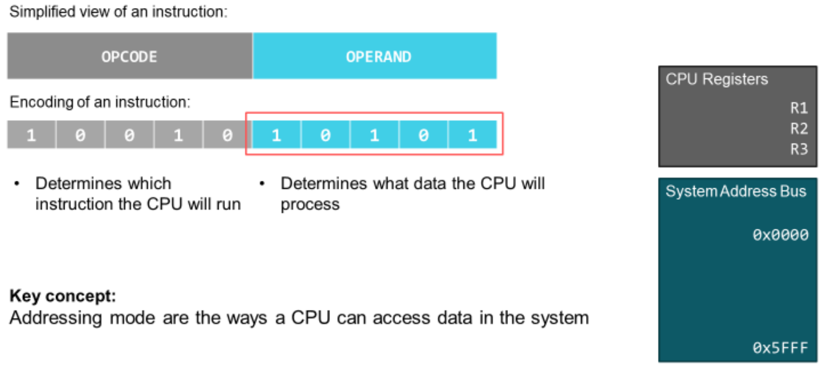
Addressing:
The address of `val` is `0x0104`.

The CPU fetches the data before it performs the compare operation. Let's say that data is `0x32`

6

Texas Instruments

CPU addressing modes



- Determines which instruction the CPU will run
- Determines what data the CPU will process

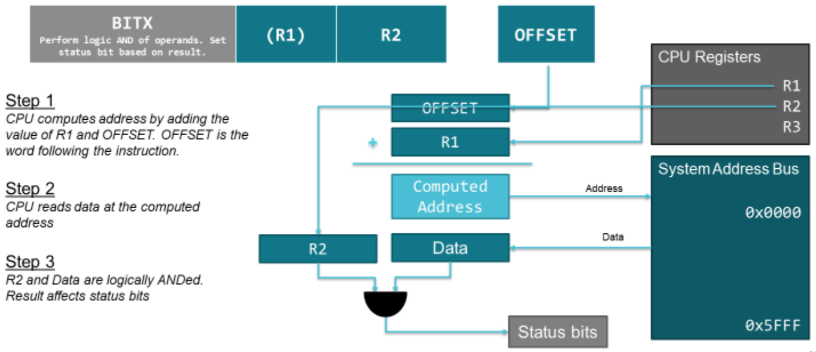
Key concept:
Addressing mode are the ways a CPU can access data in the system

7

Texas Instruments

Indexed addressing mode

→ Data at address in the register plus an offset



Step 1
CPU computes address by adding the value of R1 and OFFSET. OFFSET is the word following the instruction.

Step 2
CPU reads data at the computed address

Step 3
R2 and Data are logically ANDed. Result affects status bits

11

Texas Instruments

Addressing modes in practice

C Code:

```
int is_odd(int *val) {
    if (*val & 0x1) {
        return 1;
    } else
        return 0;
}
```

Assembly:

```
DECD.W  SP
MOV.W   R12,0x0000(SP)
MOV.W   #1,R15
BIT.W   @R12,R15
JEQ     ($C$L1)
MOV.W   #1,R12
JMP     ($C$L2)
CLR.W   R12
INCD.W  SP
RETA
```

Description
Test if bits are set in both @R12 and R15
Source operand addressing mode
Register indirect
Destination addressing mode
Register

15

Texas Instruments

To find more microcontrollers technical resources and search products, visit **ti.com/microcontrollers**.