COMPUTER STRUCTURE (II)

Computer Architecture and Operating Systems

Instruction set

- Instruction types

- Addressing modes

Instruction types

- Data movement instructions
- Arithmetic and logical instructions
- Control flow instruction

Control flow instructions

Unconditional jump:
  jmp address
  PC <- address

Conditional jump:
  jne offset (jge, jg, je, jle, jl, jc, …)
  PC <- (PC) + offset

  Usually, offset is a 8 bit value

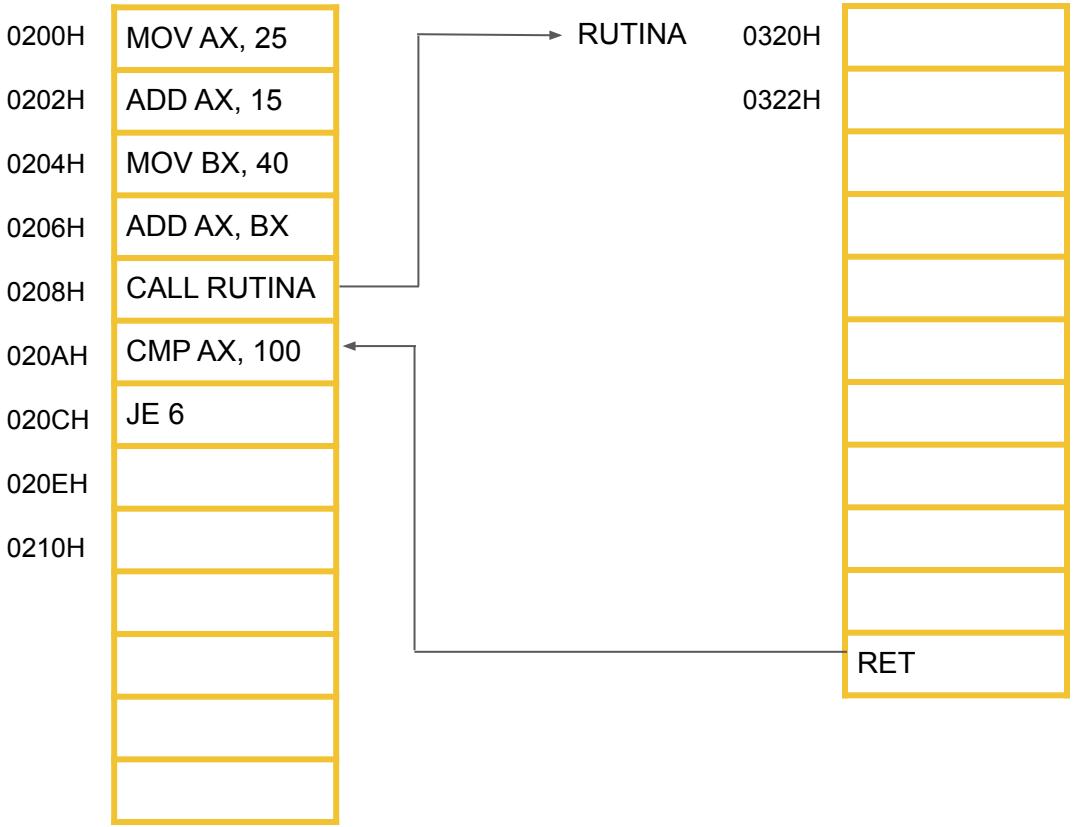
Jumps with return

RISC-V

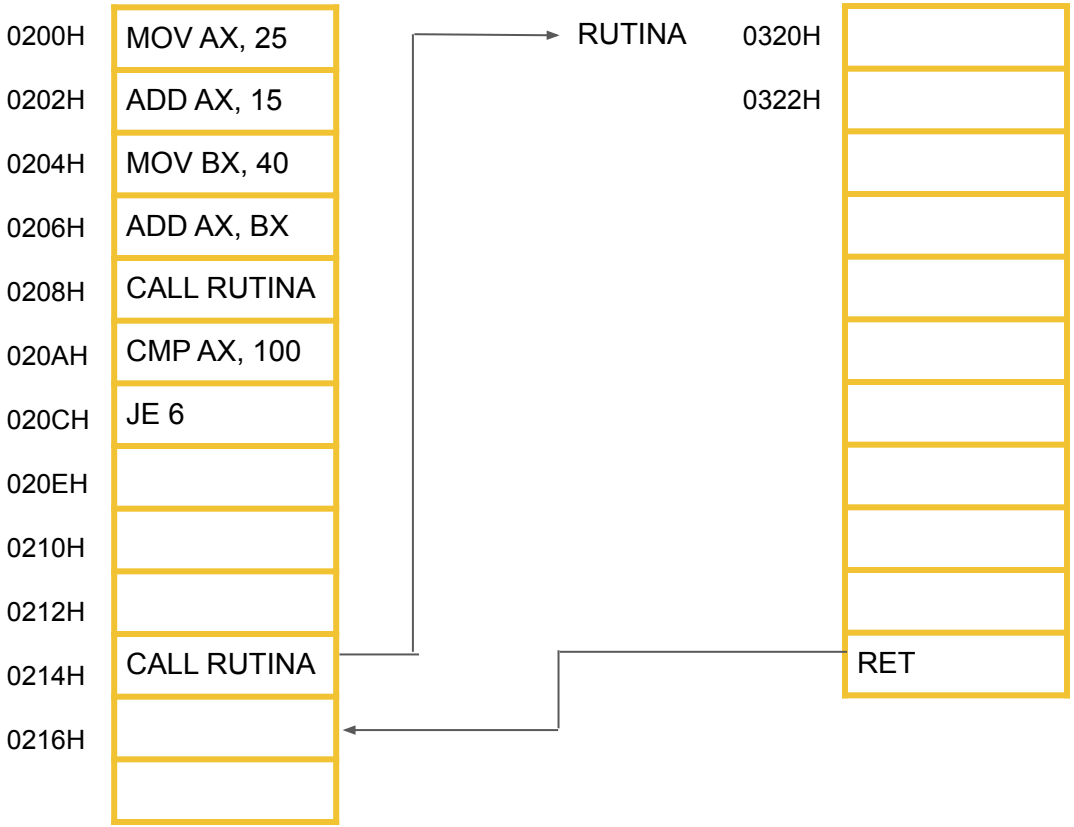
Conditional jump:
  beq rs1, rs2, imm

  bne, blt, bge, …

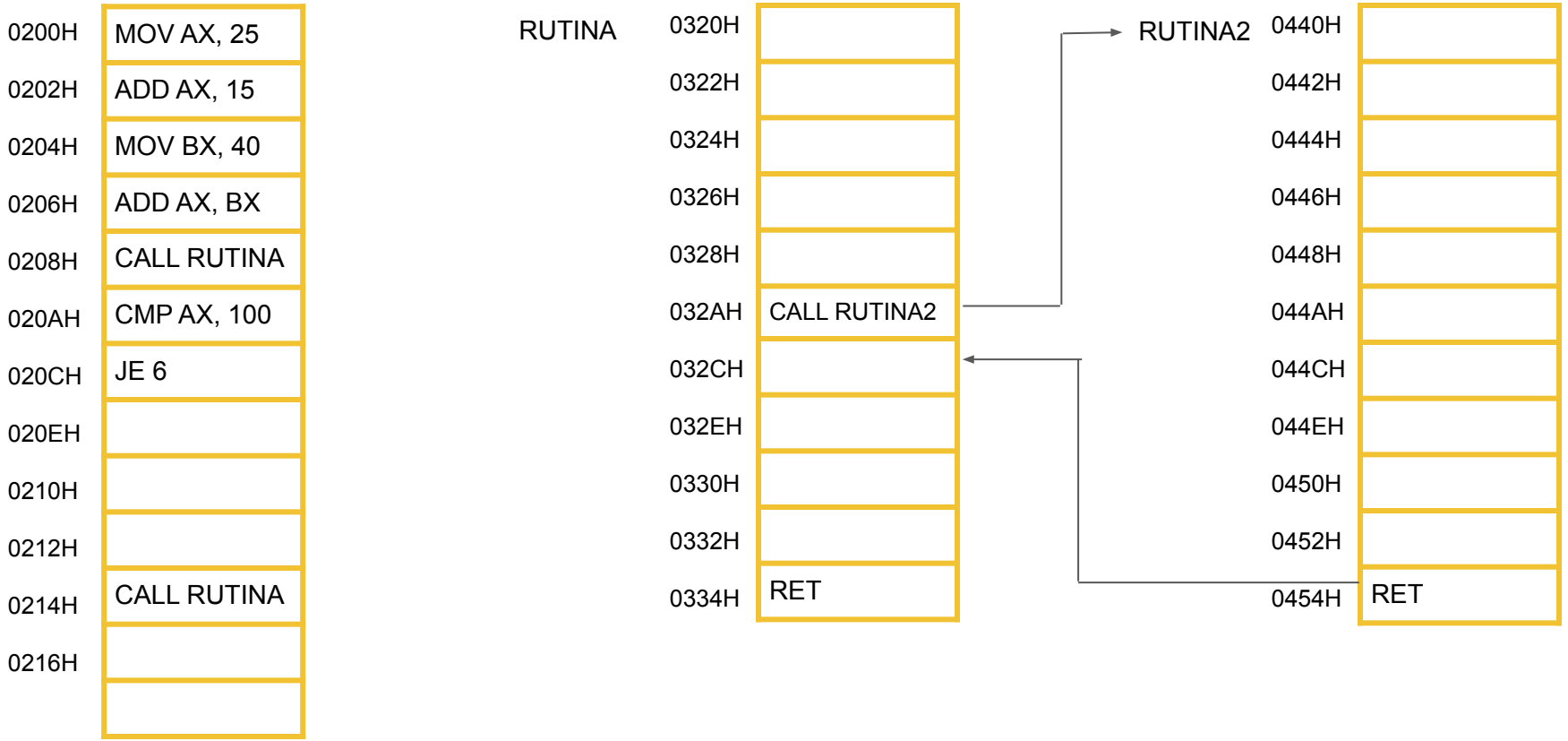
Jumps with return



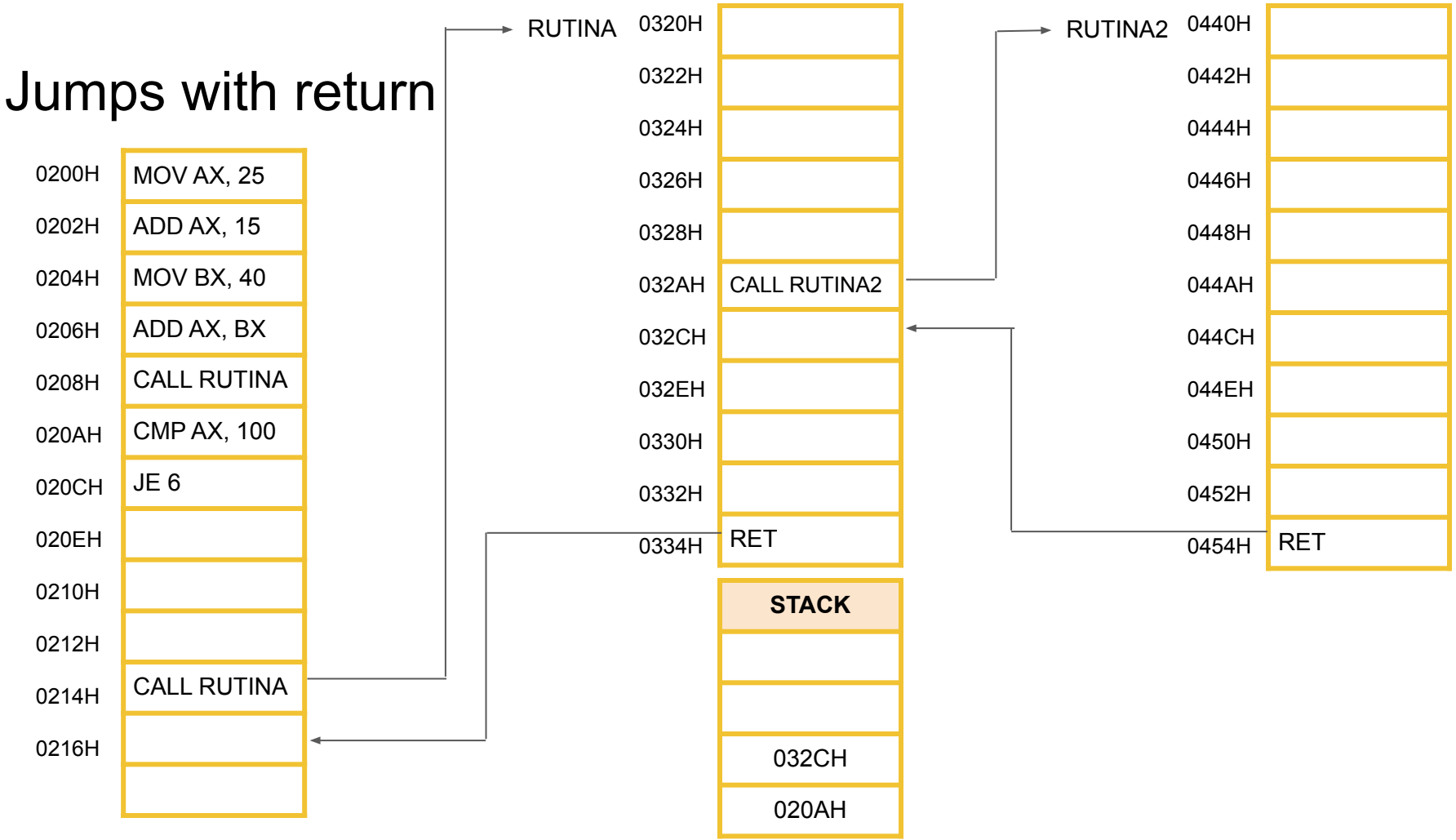
Jumps with return



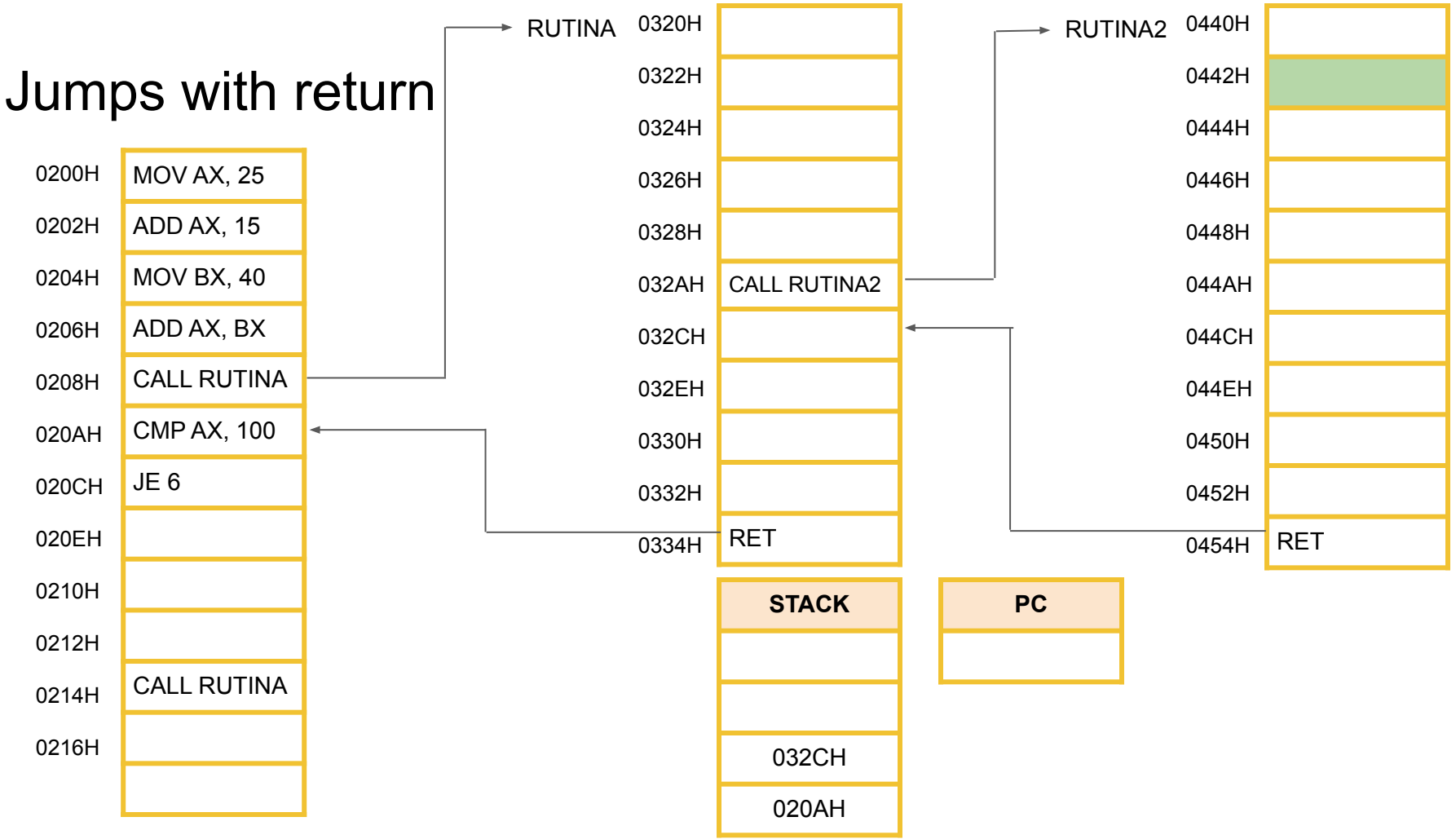
Jumps with return



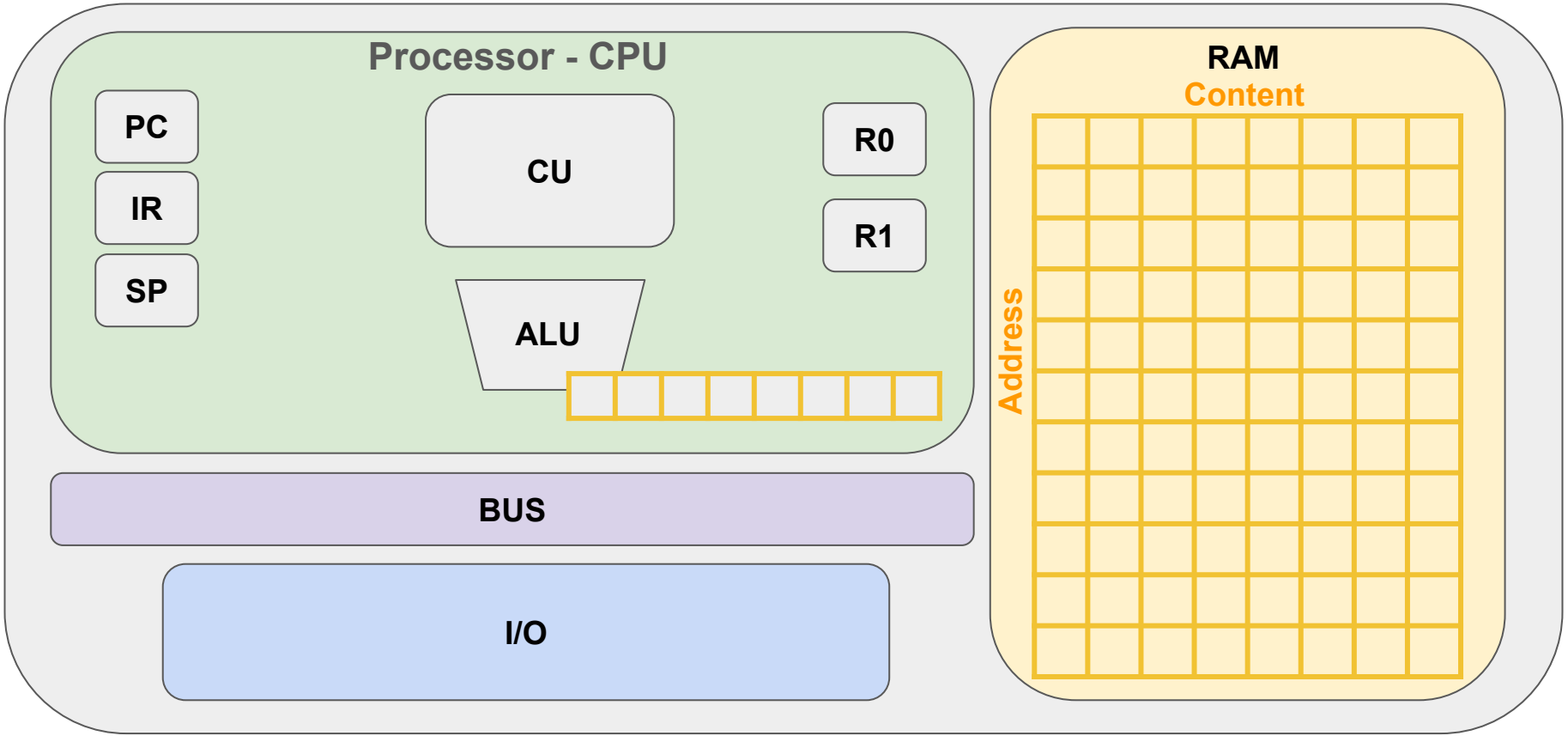
Jumps with return



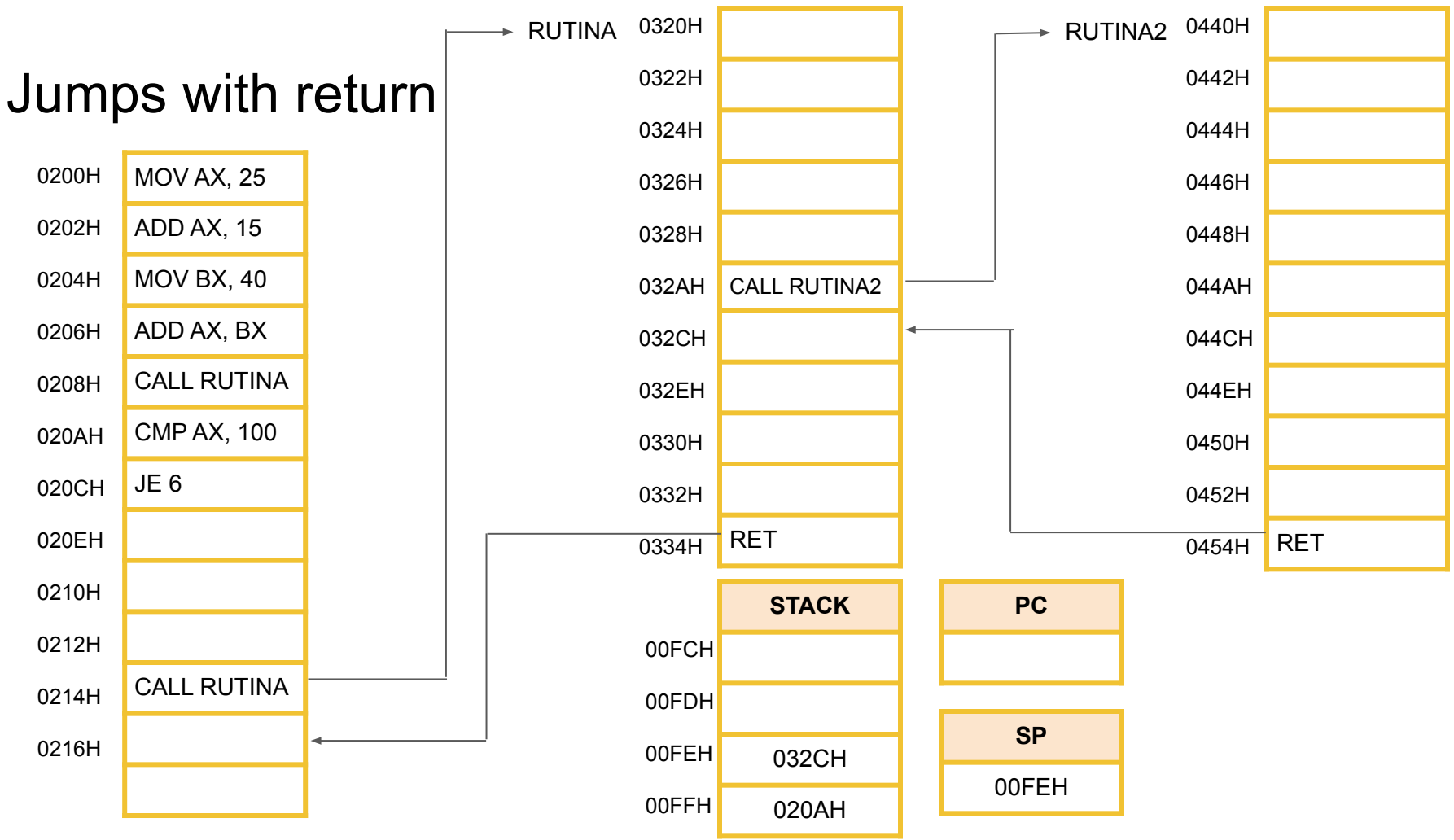
Jumps with return



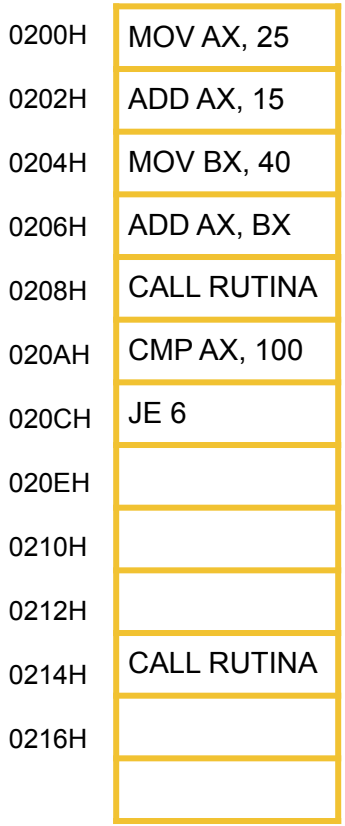
The computer

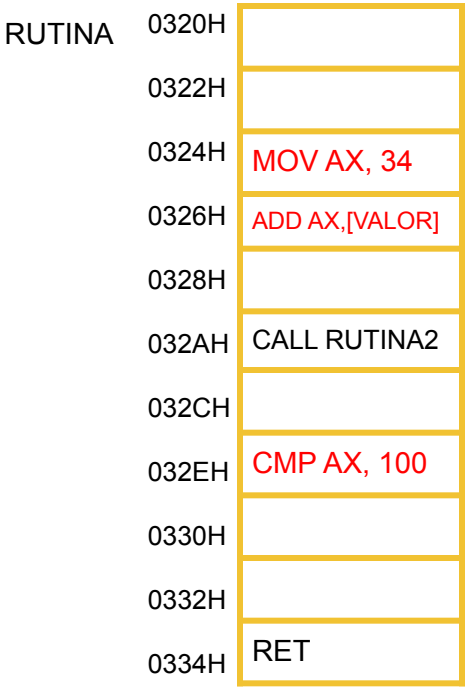


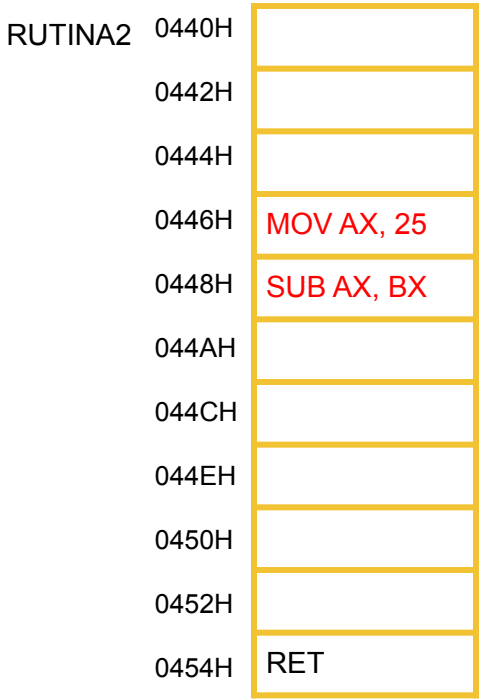
Jumps with return



Jumps with return







Jumps with return

0200H	MOV AX, 25
0202H	ADD AX, 15
0204H	MOV BX, 40
0206H	ADD AX, BX
0208H	CALL RUTINA
020AH	CMP AX, 100
020CH	JE 6
020EH	
0210H	
0212H	
0214H	CALL RUTINA
0216H	

RUTINA	0320H	PUSH AX
	0322H	PUSH BX
	0324H	MOV AX, 34
	0326H	ADD AX,[VALOR]
	0328H	
	032AH	CALL RUTINA2
	032CH	
	032EH	CMP AX, 100
	0330H	POP BX
	0332H	POP AX
	0334H	RET

RUTINA2	0440H	PUSH AX
	0442H	PUSH BX
	0444H	
	0446H	MOV AX, 25
	0448H	SUB AX, BX
	044AH	
	044CH	
	044EH	
	0450H	POP BX
	0452H	POP AX
	0454H	RET

Instruction types

- Data movement instructions
  - PUSH: Store at stack
  - POP: Retrieve from stack

  - LEA: Load Effective Address

    LEA BX, [VALUE]

Instruction set

- Instruction types
  - Programming

- **Addressing modes**
  - Data structures
  - Programming

- Instruction format

Addressing modes

- Operand types
  - Immediates

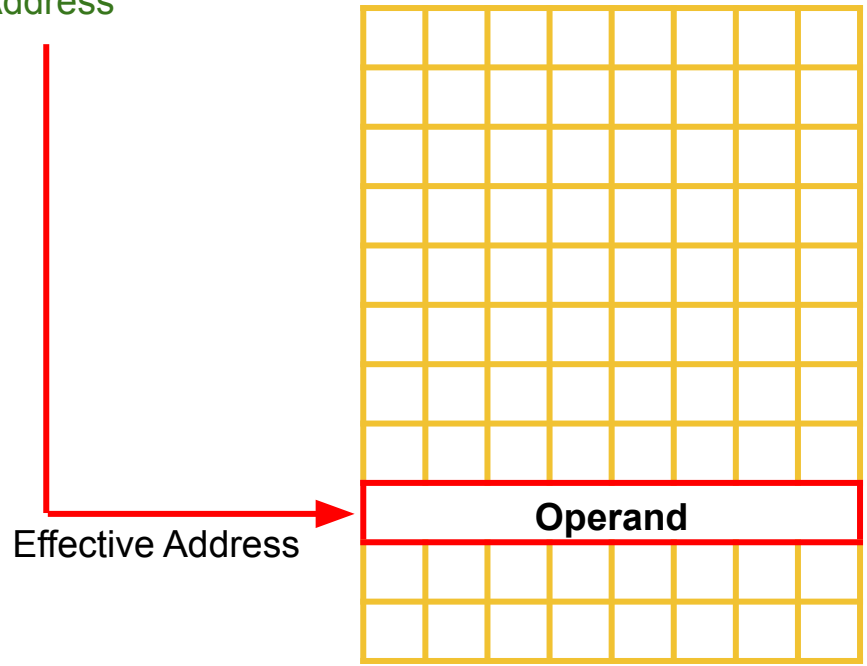
    MOV AX, 123FH LDA #32

  - Registers
  - **Memory**

Addressing modes - Direct addressing

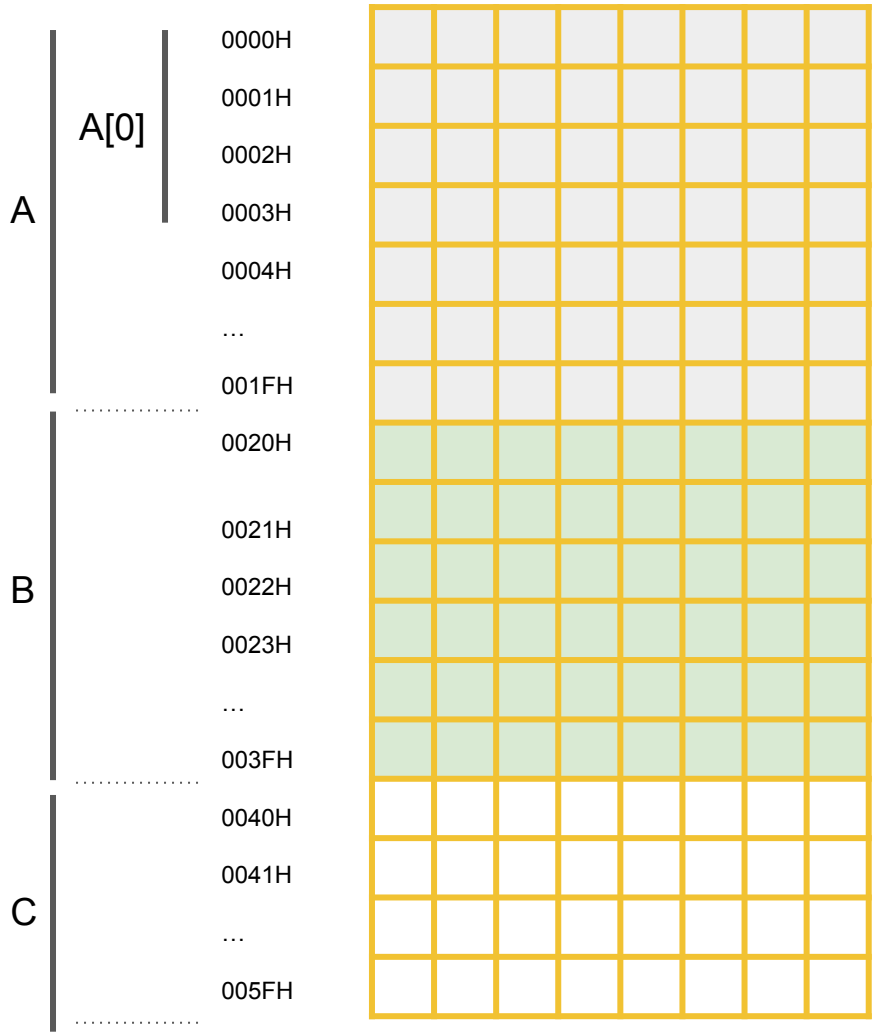
MOV AX, [0324H]

LDA 0200H



Direct addressing

```
int A[8], B[8], C[8];
int i;
for(i=0; i<8; i++) {
  C[i] = A[i] + B[i]
}
```



Direct addressing

MOV EAX, [0000H]

ADD EAX, [0020H]

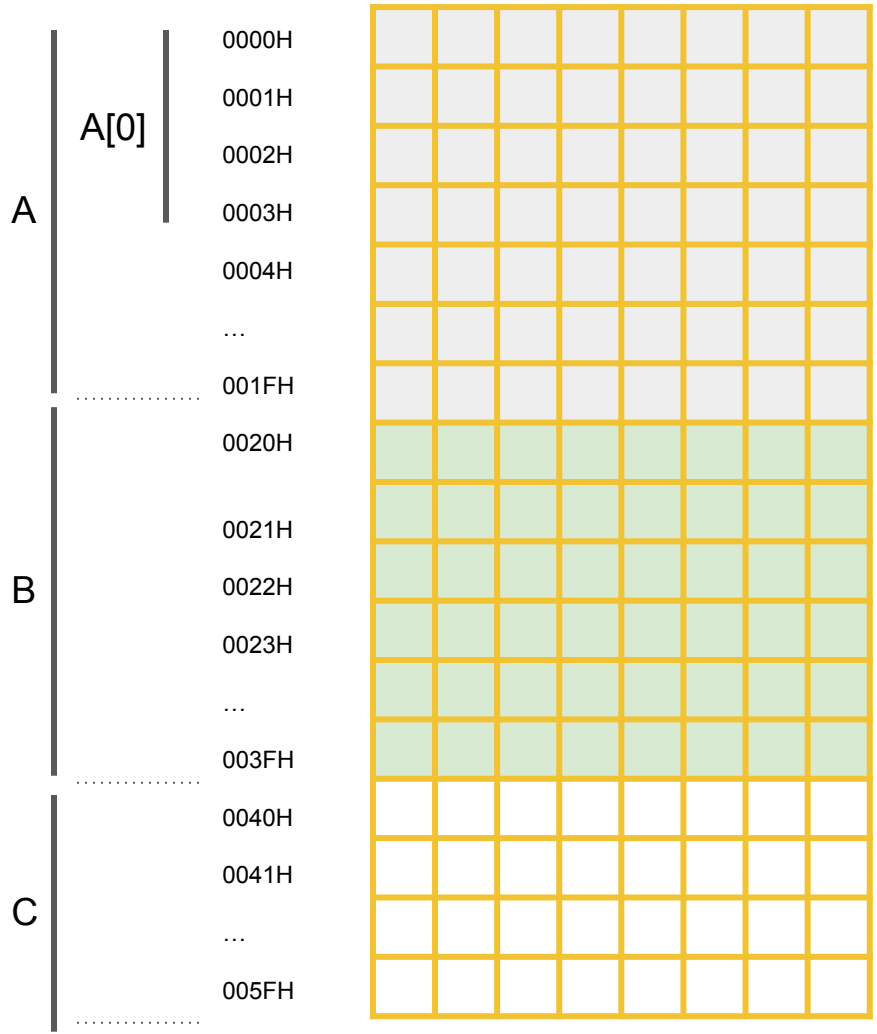
MOV [0040], EAX

MOV EAX, [0004H]

ADD EAX, [0024H]

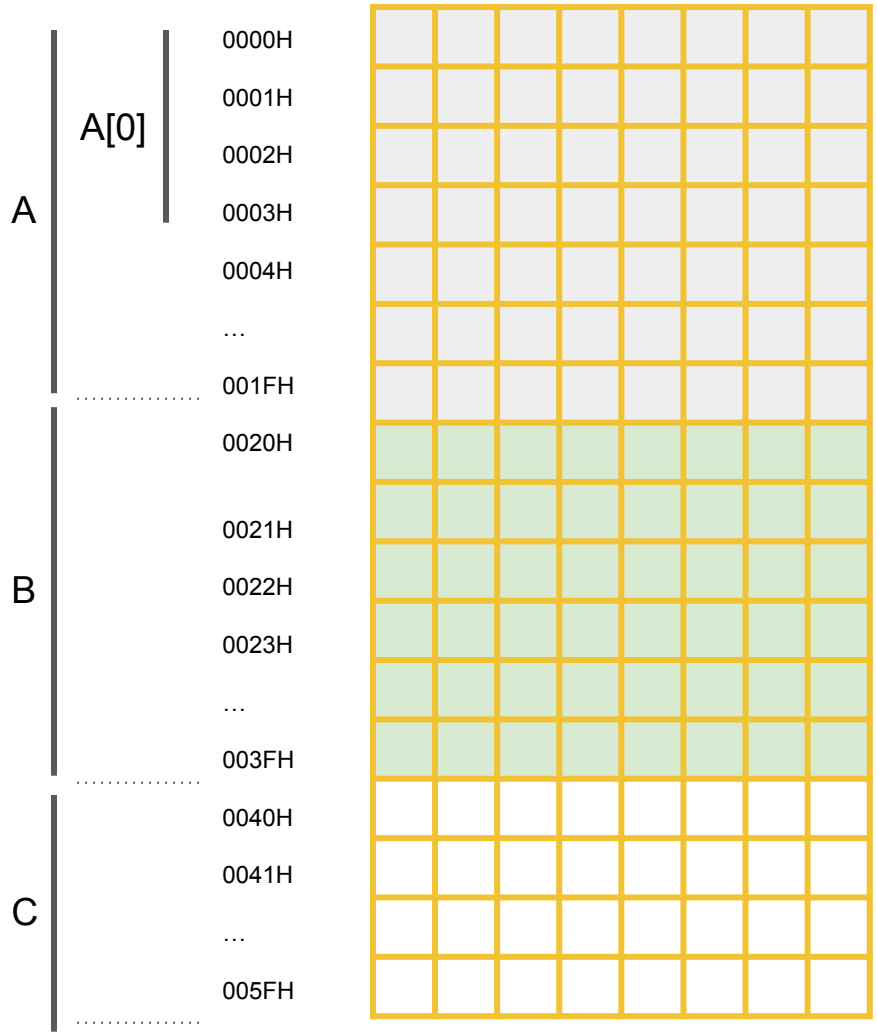
MOV [0044], EAX

...

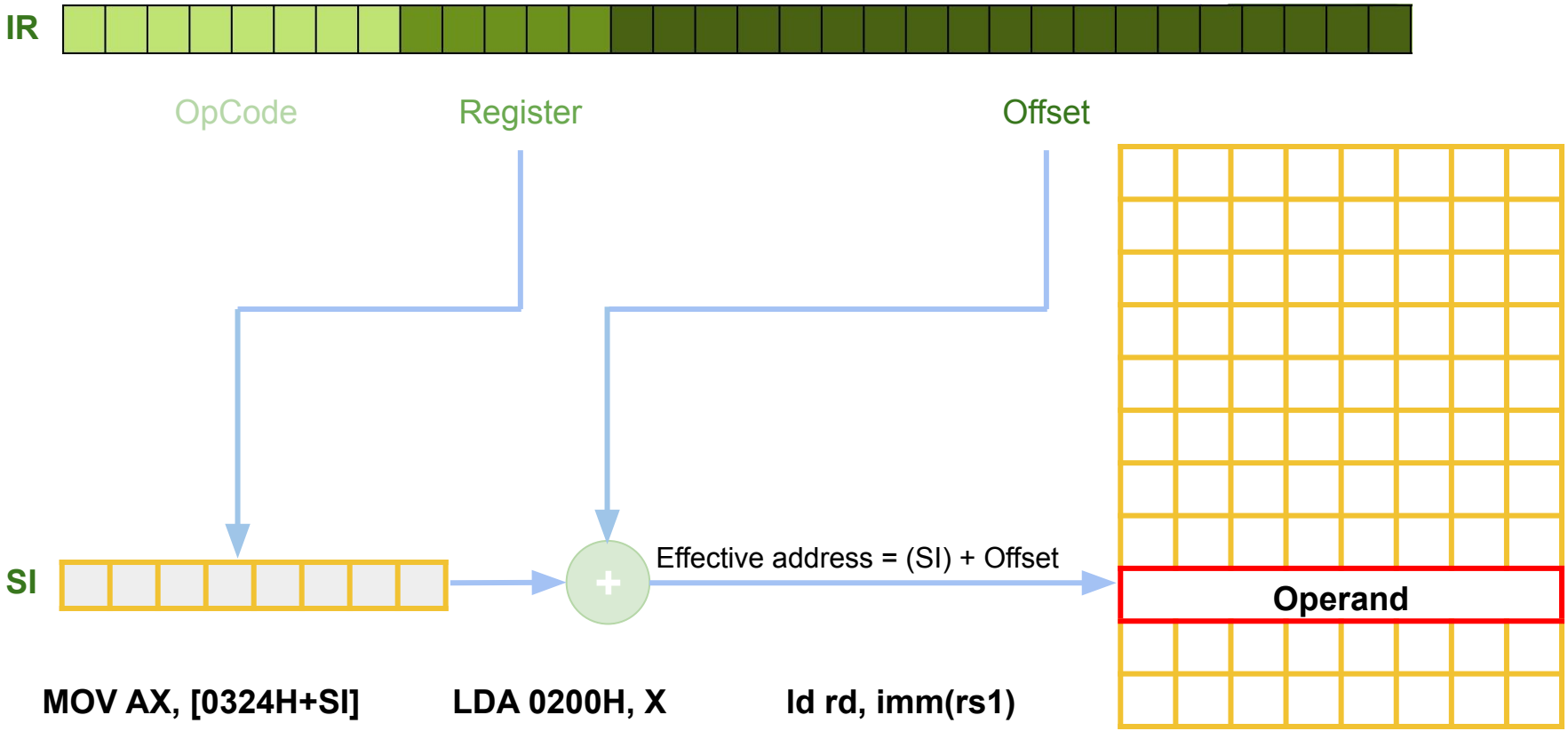


Addressing mode???

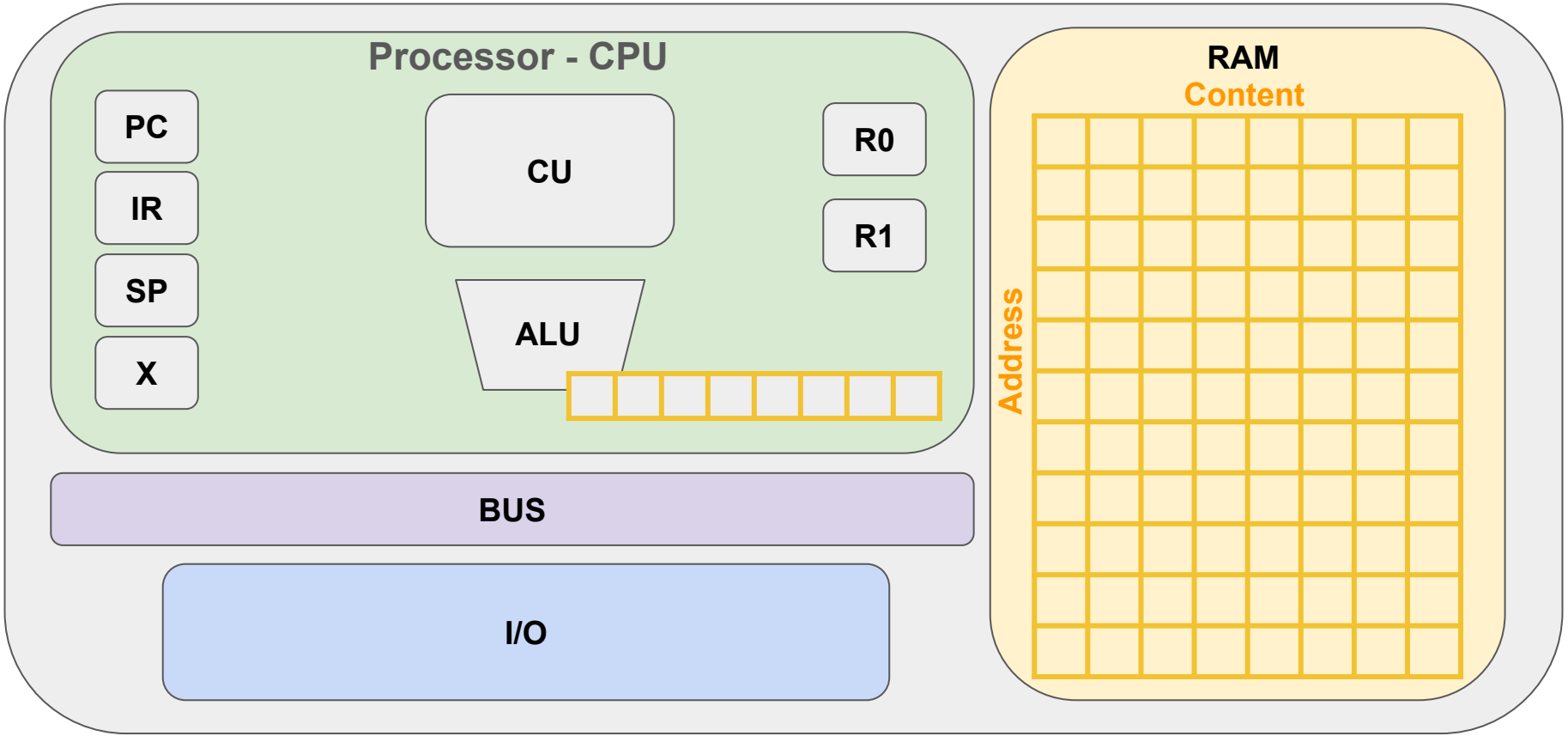
```
        MOV ESI, 0
BUCLE:  MOV EAX, [A+ESI]
        ADD EAX, [B+ESI]
        MOV [C+ESI], EAX
        ADD ESI, 4
        CMP ESI, 32
        JL BUCLE
```



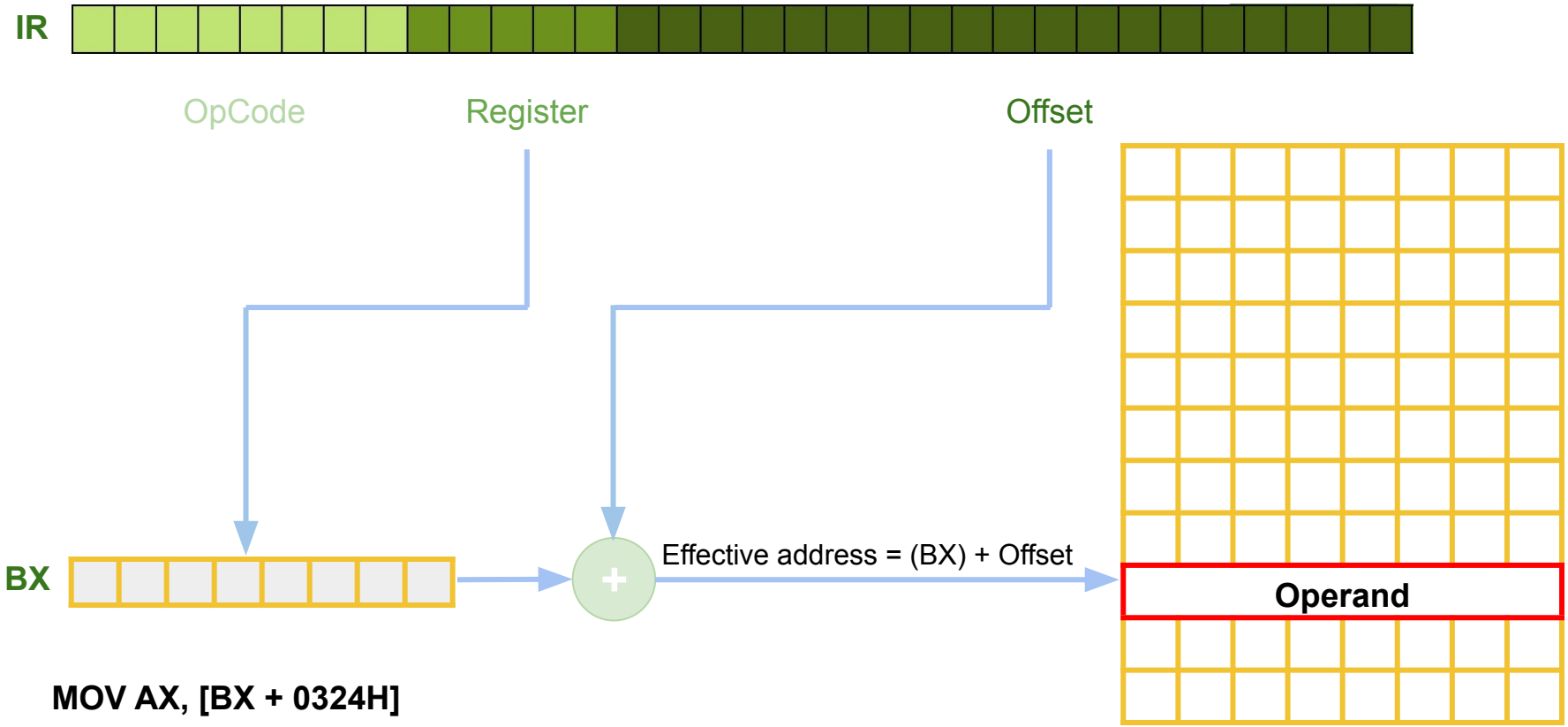
Addressing modes - Indexed addressing



The computer

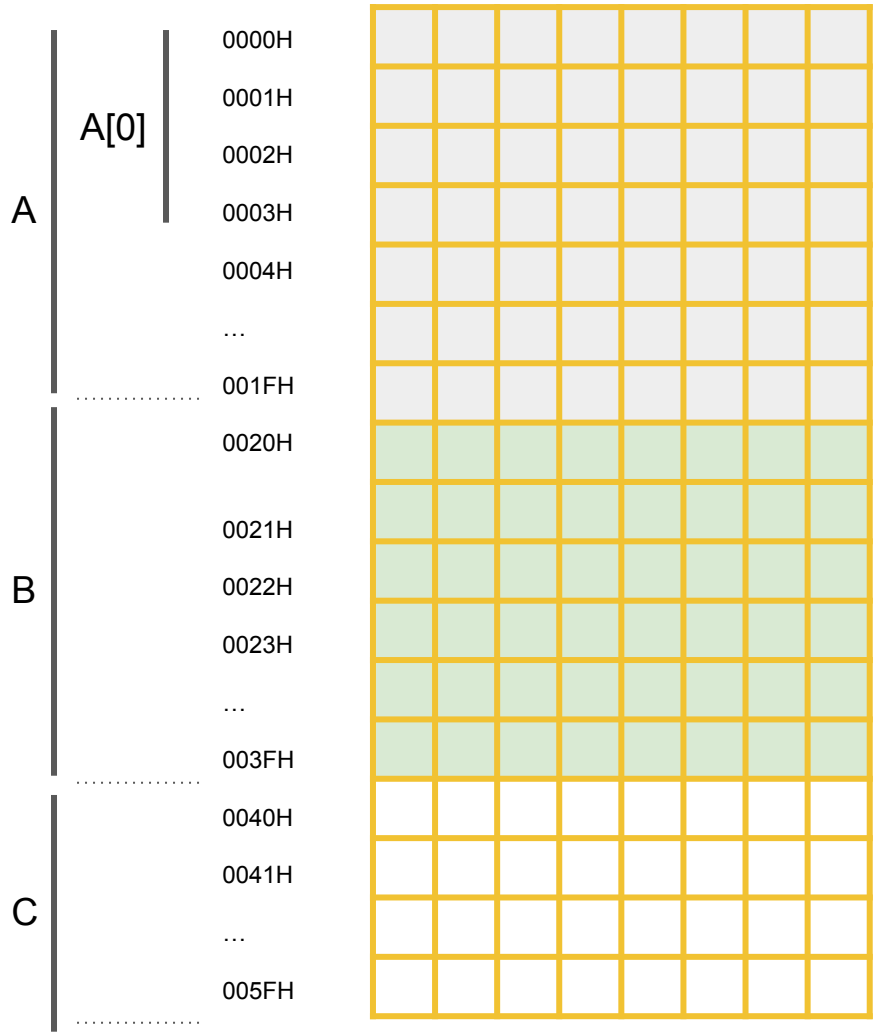


Addressing modes - Base register addressing

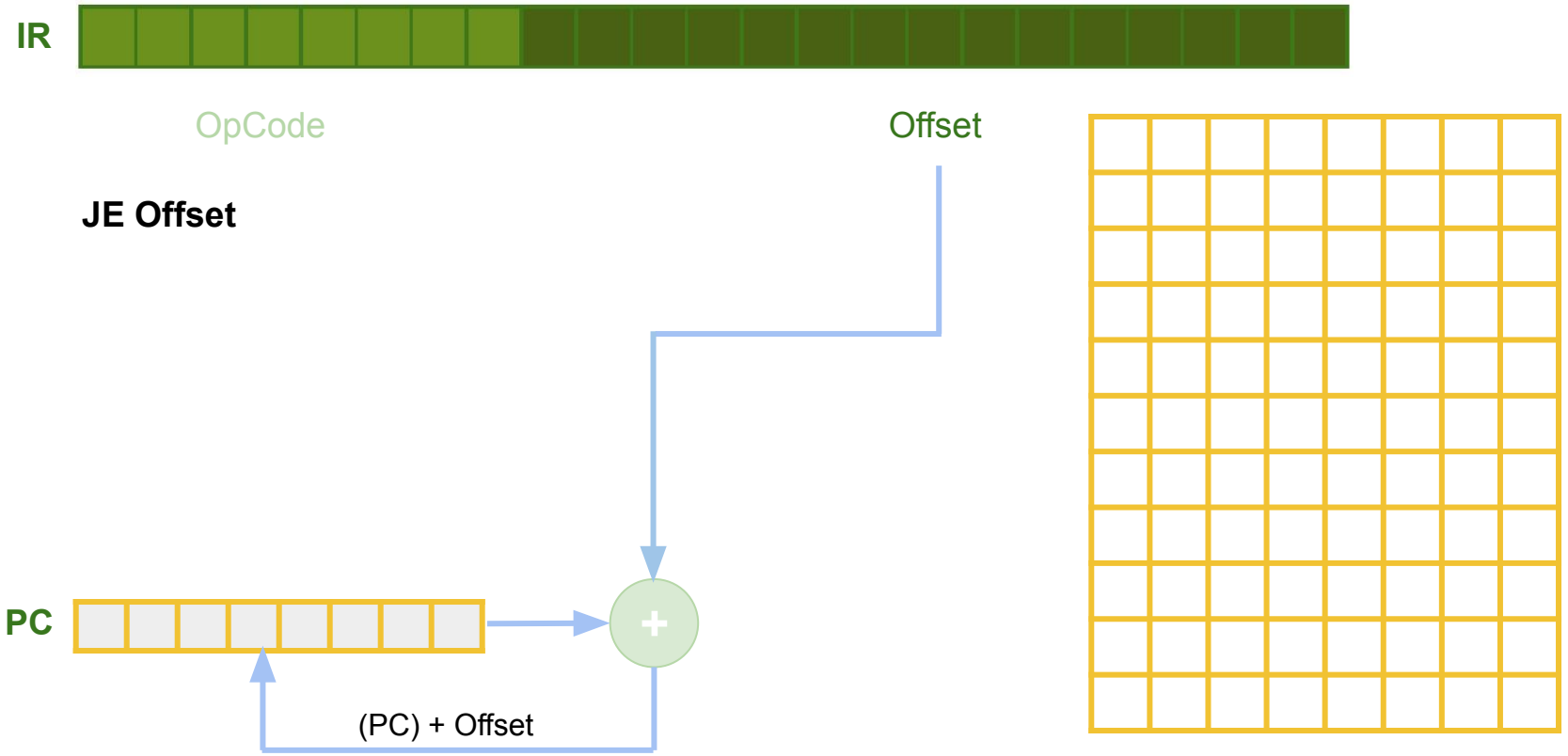


Base register addressing

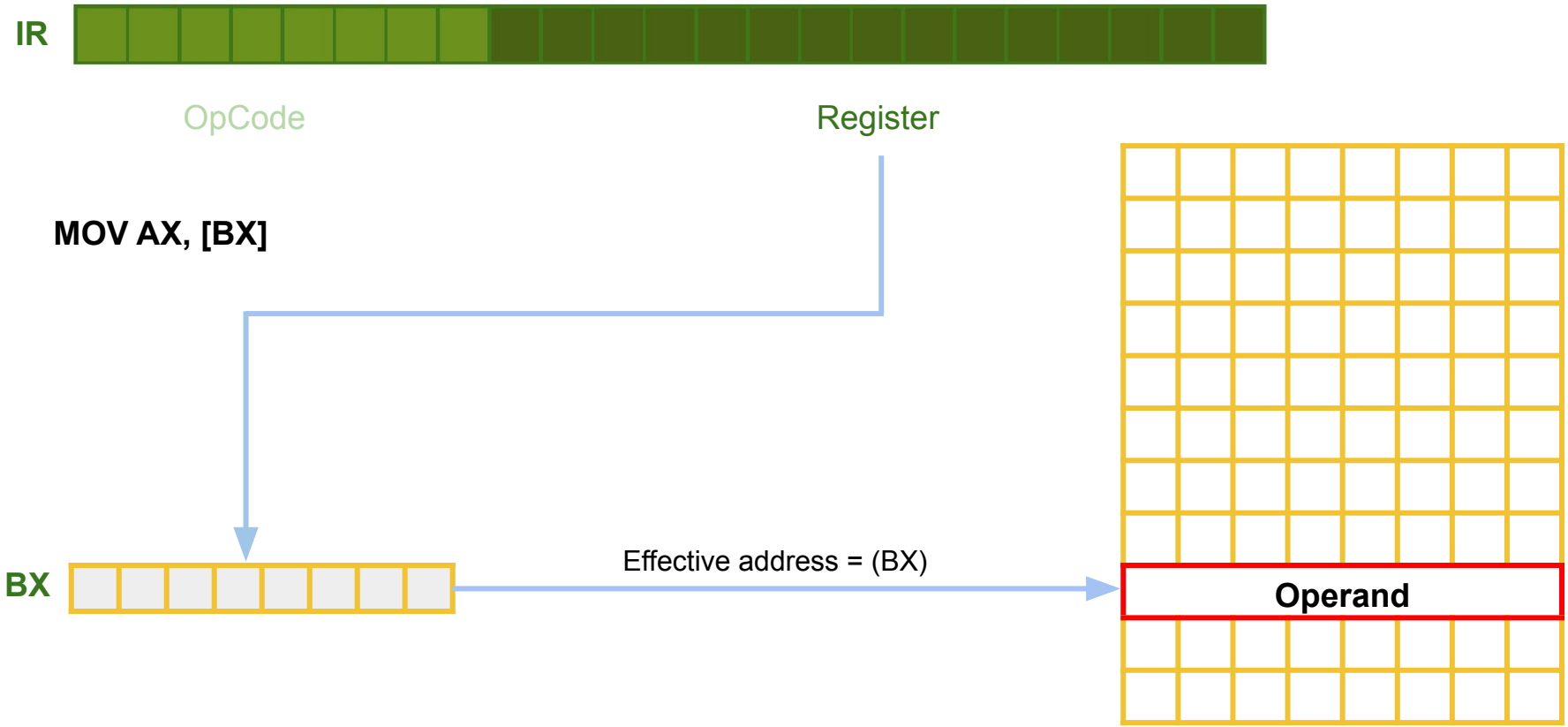
```
        LEA EBX, [A]
BUCLE:  MOV EAX, [EBX+0]
        ADD EAX, [EBX+20H]
        MOV [EBX+40H], EAX
        ADD EBX, 4
        CMP EBX, 32
        JL BUCLE
```



Addressing modes - PC relative addressing



Addressing modes - Register indirect addressing

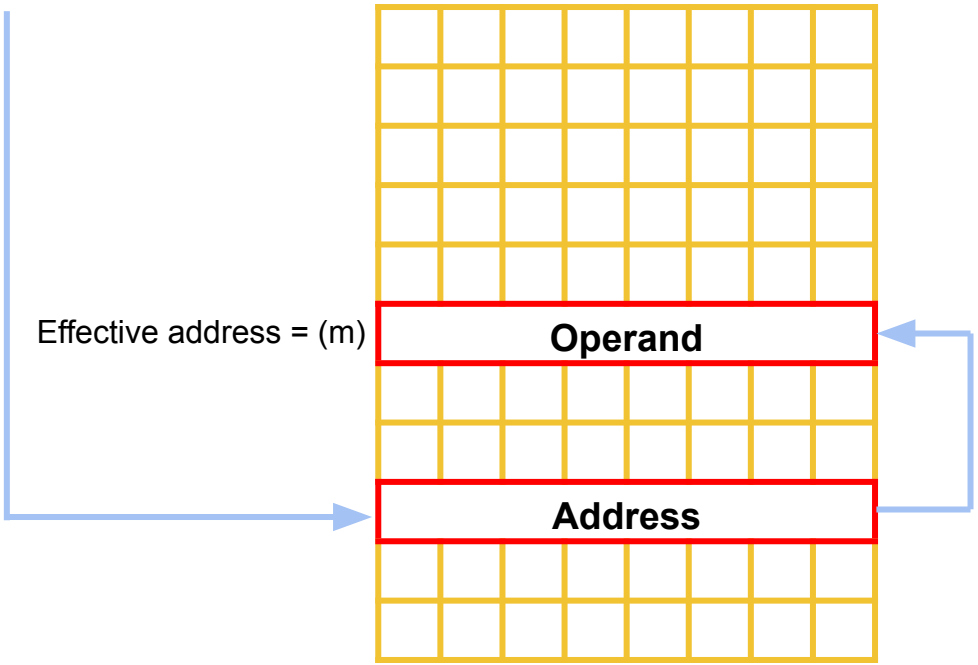


Addressing modes - Memory indirect addressing

OpCode

Address

LDA (0200H)



Addressing modes

Operand types:

- Immediates
- Registers
- Memory
  - Direct
  - Indexed
  - Base register
  - PC relative
  - Register indirect
  - Memory indirect

Exercise 1

We have an array of 20 elements of integer type of 32 bits. The array starts at the logic address called Vector. Fill the gaps in the assembler code on which we have in the **Vegades** variable the number of times that the value stored in variable **Valor** is found.

Valor = 33

12 **33** 56 15 **33** 67 23 23 89 **33** 19 87 **33** 11 56 **33** 98 23 88 12

Vegades = 5

Exercise 1

We have an array of 20 elements of integer type of 32 bits. The array starts at the logic address called Vector. Fill the gaps in the assembler code on which we have in the **Vegades** variable the number of times that the value stored in variable **Valor** is found.

```
INI:    MOV ESI, 0
        MOV ___, 0
        MOV EAX, ____
BUCLE:  CMP ESI, _____
        _____ FI
        CMP  EAX, _______
        JNE  SEG
        INC ____
SEG:    ADD  ESI, ____
        ___ BUCLE
FI:     MOV ___, EBX
```

Valor = 33

12 **33** 56 15 **33** 67 23 23 89 **33** 19 87 **33** 11 56 **33** 98 23 88 12

Vegades = 5

Exercise 1

```
INI:     MOV ESI, 0
         MOV _EBX_, 0
         MOV EAX, _[Valor]__
BUCLE:   CMP ESI, __80___
         __JGE__  FI
         CMP  EAX, _Vector[ESI]__
         JNE  SEG
         INC _EBX__
SEG:     ADD  ESI, _4__
         _JMP_ BUCLE
FI:      MOV _[Vegades]_, EBX
```

Valor = 33

12 **33** 56 15 **33** 67 23 23 89 **33** 19 87 **33** 11 56 **33** 98 23 88 12

Vegades = 5

Exercise

Having this piece of code:

1. 0200 MOV A, (2)
2. 0201 ADD A, #4
3. 0202 MOV 6, A
4. 0203 SUB A, 3[X]
5. 0204 MOV (1), X
6. 0205 MOV A, 1[X]

Where addressing modes are:

n => Direct

#n => Immediate

n[X] => Indexed

(n) => Indirect

Show in the next table register and memory positions evolution, indicating changing values.

	Registres		Memòria								
INSTRUCCIÓ	A	X	0	1	2	3	4	5	6	7	8
	0	1	6	5	4	5	3	9	2	1	7
MOV A, (2)											
ADD A, #4											
MOV 6, A											
SUB A, 3[X]											
MOV (1), X											
MOV A, 1[X]											

Exercise

Having this piece of code:

1. 0200 MOV A, (2)
2. 0201 ADD A, #4
3. 0202 MOV 6, A
4. 0203 SUB A, 3[X]
5. 0204 MOV (1), X
6. 0205 MOV A, 1[X]

Where addressing modes are:

n => Direct

#n => Immediate

n[X] => Indexed

(n) => Indirect

Show in the next table register and memory positions evolution, indicating changing values.

	Registres		Memòria								
INSTRUCCIÓ	A	X	0	1	2	3	4	5	6	7	8
	0	1	6	5	4	5	3	9	2	1	7
MOV A, (2)	3						Rd				
ADD A, #4	7										
MOV 6, A									7		
SUB A, 3[X]	4						Rd				
MOV (1), X				Rd				1			
MOV A, 1[X]	4				Rd						

Exercise

	Registres		Memòria								
INSTRUCCIÓ	A	X	0	1	2	3	4	5	6	7	8
	0	1	6	5	4	5	3	9	2	1	7
MOV A, (2)	3						Rd				
ADD A, #4	7										
MOV 6, A									7		
SUB A, 3[X]	4						Rd				
MOV (1), X				Rd				1			
MOV A, 1[X]	4				Rd						

COMPUTER STRUCTURE (II)

Computer Architecture and Operating Systems