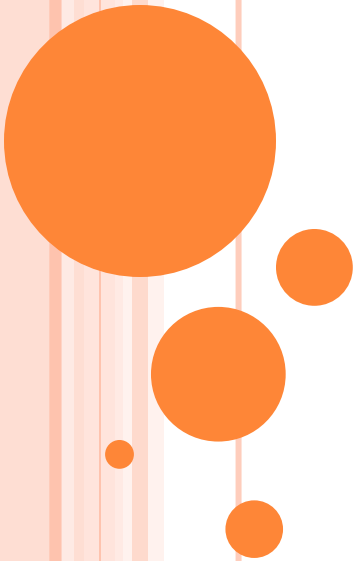


Module 3

Instruction Set Architectures

Instruction Set Architecture

Encompasses all the information necessary to write a machine language program that will run correctly, including instructions, registers, memory access, I/O, and so on.

The words of a computer's language are called *instructions*, and its vocabulary is called an **instruction set**.

Different Types of Architecture Classification

- RISC and CISC
- Accumulator based ,Stack based, General purpose registers

INSTRUCTION TYPES- FORMAT OF INSTRUCTIONS

- INSTRUCTION TYPES refer to the format of instructions
- Classified based on the number of operands used in the instruction
- The classification includes:
  - Three address instructions
  - Two address instructions
  - One address instructions
  - Zero address instructions
- USES THREE TYPES OF ARCH.

1.STACK

2.ACCUMULATOR

3.GENERAL PURPOSE REGISTERS

Three Address Instructions

- C=A+B
- C ← [A]+[B]
- We will use the variable names to the corresponding memory location
- ADD A, B, C

  OPERATION SOP1 SOP2 DEST OP

- A AND B ARE SOURCE ,C IS DESTINATION
- CONTENTS OF A AND B ARE ADDED UP AND STORED IN C
- If 32 bit of computer is used then to perform addition operation we require 3k bits and it is too large to fit in one word

- ADD A,B
- A IS SOURCE ,B IS ACTING AS SOURCE AND DEST.

MOVE B,C

ADD A,C

NOW THE VALUE OF B IS UNCHANGED

- If 32 bit of computer is used then to perform addition operation we require 2k bits and it is too large to fit in one word

ONE ADDRESS INSTRUCTION

- IMPLICIT LOCATION VIA ACCUMULATOR

  Load A ---------- moves the contents of A to Accumulator (Implied)

  Add B ---------- A and B gets added up

  Store C ---------- From Accumulator to memory

- USE of General Purpose Registers

  Add R_i , R_j

  Add R_i , R_j , R_k

  Move A, R_i same as Load A, R_i

  Move R_i, A same as Store R_i, A

- Number of bits needed to specify a register is much less than that of a memory location, Eg., 5 bits are required for 32 registers
- Register usage allows faster processing

INSTRUCTION TYPES

C=A+B

Using Operands from processor registers:

Move A,Ri

Move B,Rj

Add Ri,Rj

Move Rj,C

……

Using One Operand from memory:

Move A,Ri

Add B,Ri

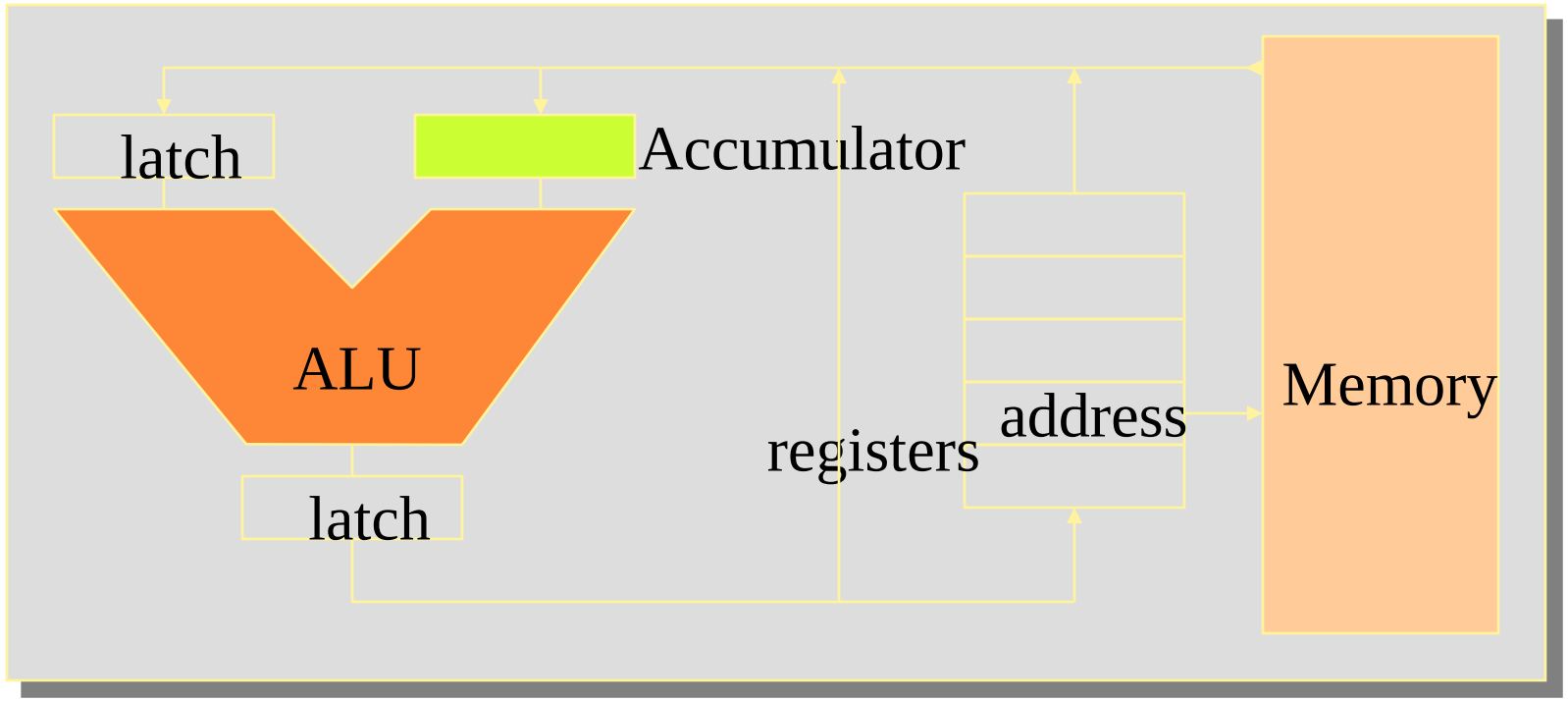
Move Ri,C

INSTRUCTION TYPES

ZERO ADDRESS INSTRUCTION

PUSH A
PUSH B
ADD
POP C

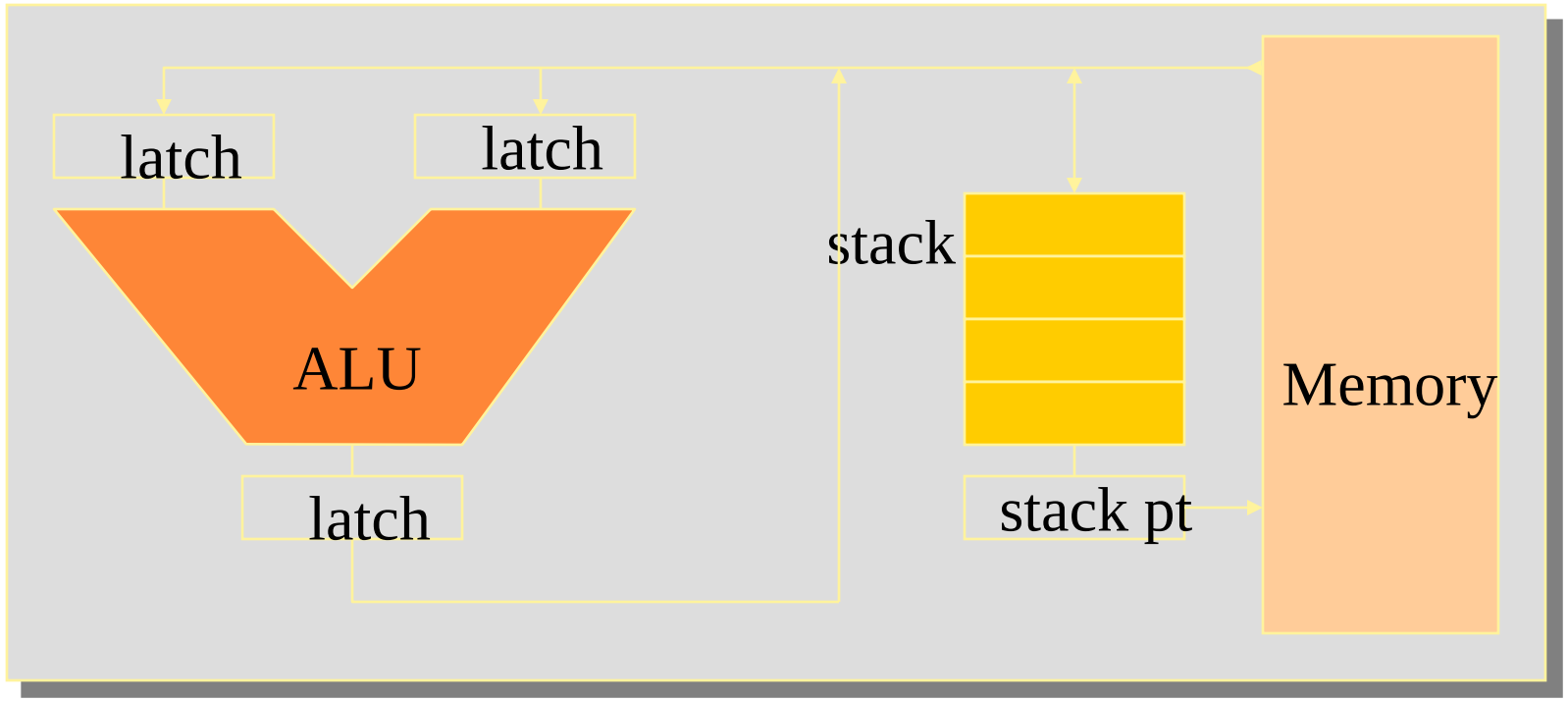
Accumulator architecture



Example code: `a = b+c;`

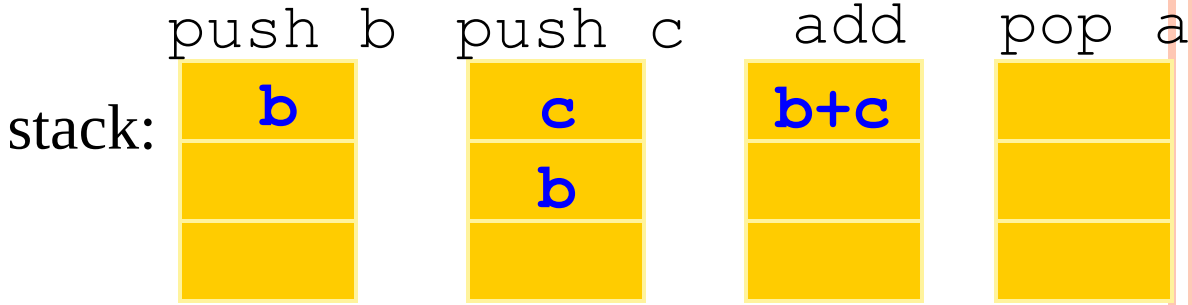
```
load  b;     // accumulator is implicit operand
add   c;
store a;
```

Stack architecture



Example code: `a = b+c;`

```
push b;
push c;
add;
pop  a;
```



Other architecture styles

Let's look at the code for C = A + B

Stack Architecture	Accumulator Architecture	Register-Memory	Memory-Memory	Register (load-store)
`Push A`	`Load A`	`Load r1,A`	`Add C,B,A`	`Load r1,A`
`Push B`	`Add B`	`Add r1,B`		`Load r2,B`
`Add`	`Store C`	`Store C,r1`		`Add r3,r1,r2`
`Pop C`				`Store C,r3`

INSTRUCTION TYPES

Program to evaluate **X = (A + B) * (C + D)**

Three Address Instructions

Computers with three address instruction formats can use each address field to

specify either processor registers or memory operands

Program to evaluate **X = (A + B) * (C + D)**

ADD	R1, A, B	R1 ← M[A] + M[B]
ADD	R2, C, D	R2 ← M[C] + M[D]
MUL	X, R1, R2	M[X] ← R1 * R2

- ✓ Results in short programs
- ✓ Instruction becomes long (many bits)

Two Address Instructions

Program to evaluate **X = (A + B) * (C + D)**

MOV	R1, A	R1← M[A]
ADD	R1, B	R1 ← R1 + M[A]
MOV	R2, C	R2 ← M[C]
ADD	R2, D	R2 ← R2 + M[D]
MUL	R1, R2	R1 ← R1 * R2
MOV	X, R1	M[X] ← R1

One Address Instructions

Use an implied AC register for all data manipulation

Program to evaluate **X = (A + B) * (C + D)**

LOAD	A	AC ← M[A]
ADD	B	AC ← AC + M[B]
STORE	T	M[T] ← AC
LOAD	C	AC ← M[C]
ADD	D	AC ← AC + M[D]
MUL	T	AC ← AC * M[T]
STORE	X	M[X] ← AC

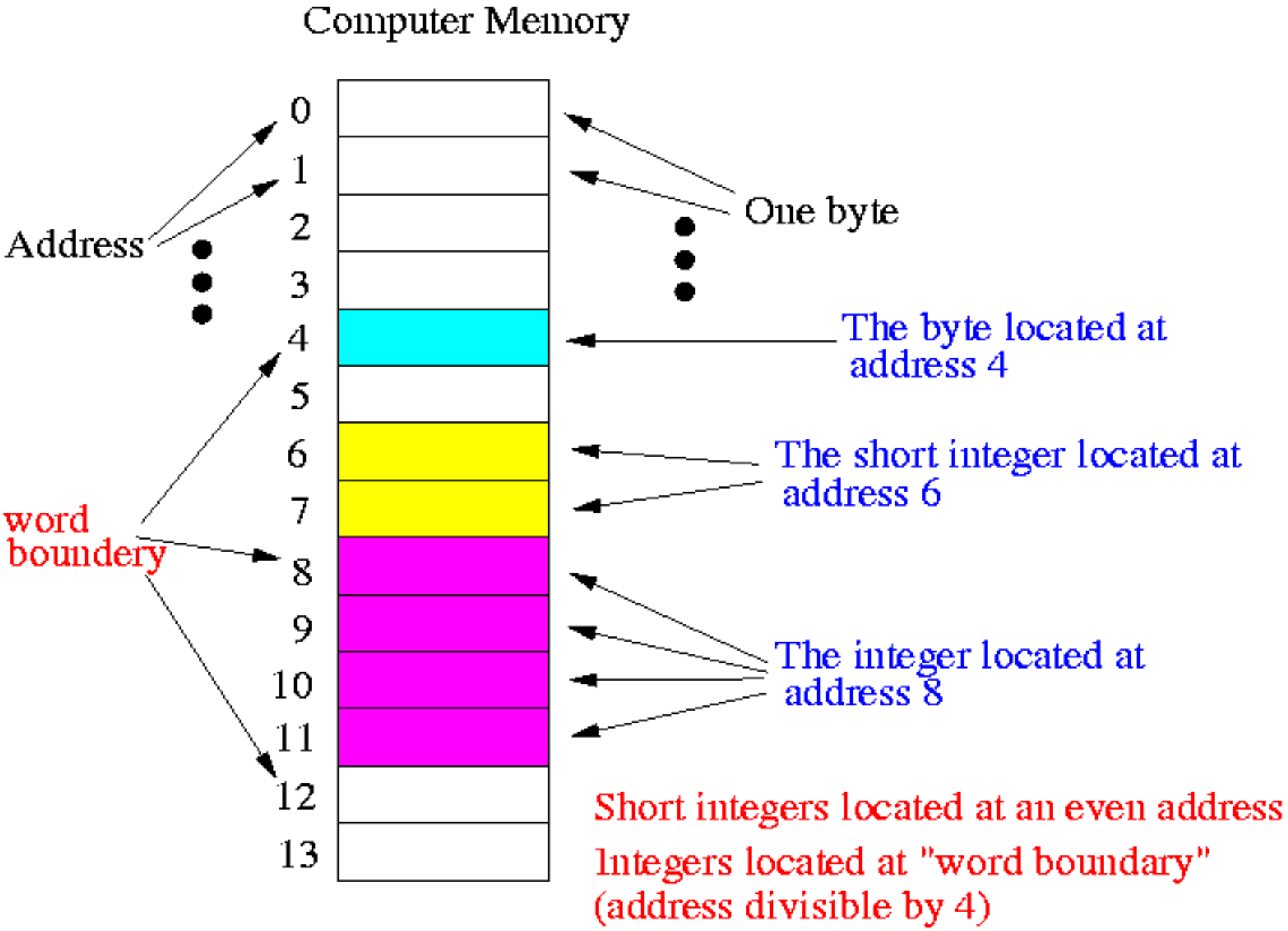
Zero Address Instructions

- ✓ Can be found in a stack-organized computer
- ✓ No need of address field for ADD and MUL instruction
- ✓ PUSH and POP requires address field to specify the operand that communicate with the stack

Program to evaluate **X = (A + B) * (C + D)**

PUSH A	TOS ← A
PUSH B	TOS ← B
ADD	TOS ← (A + B)
PUSH C	TOS ← C
PUSH D	TOS ← D
ADD	TOS ← (C + D)
MUL	TOS ← (C + D) * (A + B)
POP X	M[X] ← TOS

Note: To evaluate arithmetic expression in a stack computer, it is necessary to convert the expression into reverse polish notation .


Computer Memory
0
1
2
3
4
5
6
7
8
9
10
11
12
13
Address
word
boundery
One byte
The byte located at
address 4
The short integer located at
address 6
The integer located at
address 8
Short integers located at an even address
Integers located at "word boundary"
(address divisible by 4)