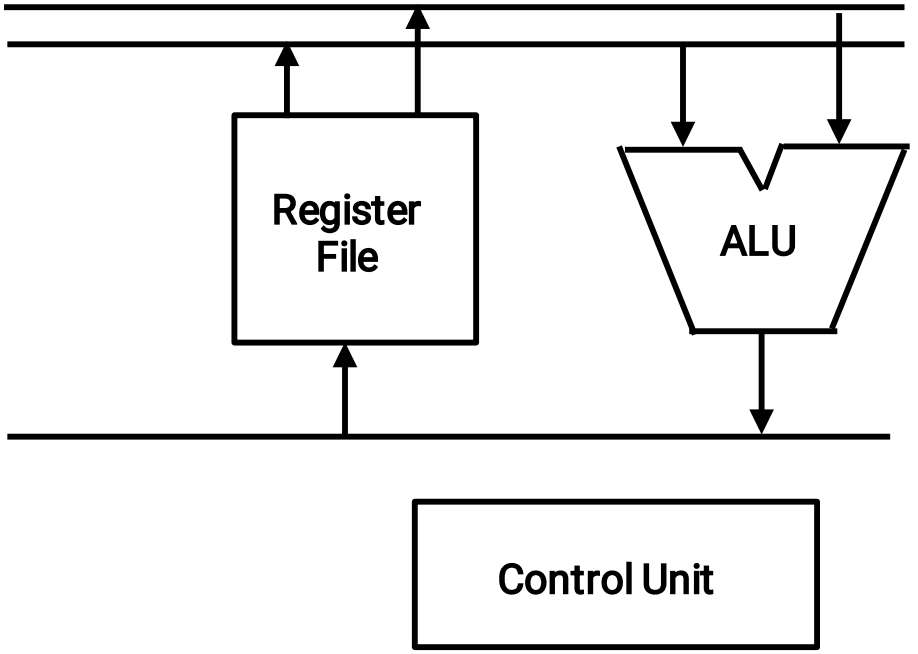
CENTRAL PROCESSING UNIT

- Introduction
- General Register Organization
- Stack Organization
- Instruction Formats
- Addressing Modes
- Reduced Instruction Set Computer

MAJOR COMPONENTS OF CPU

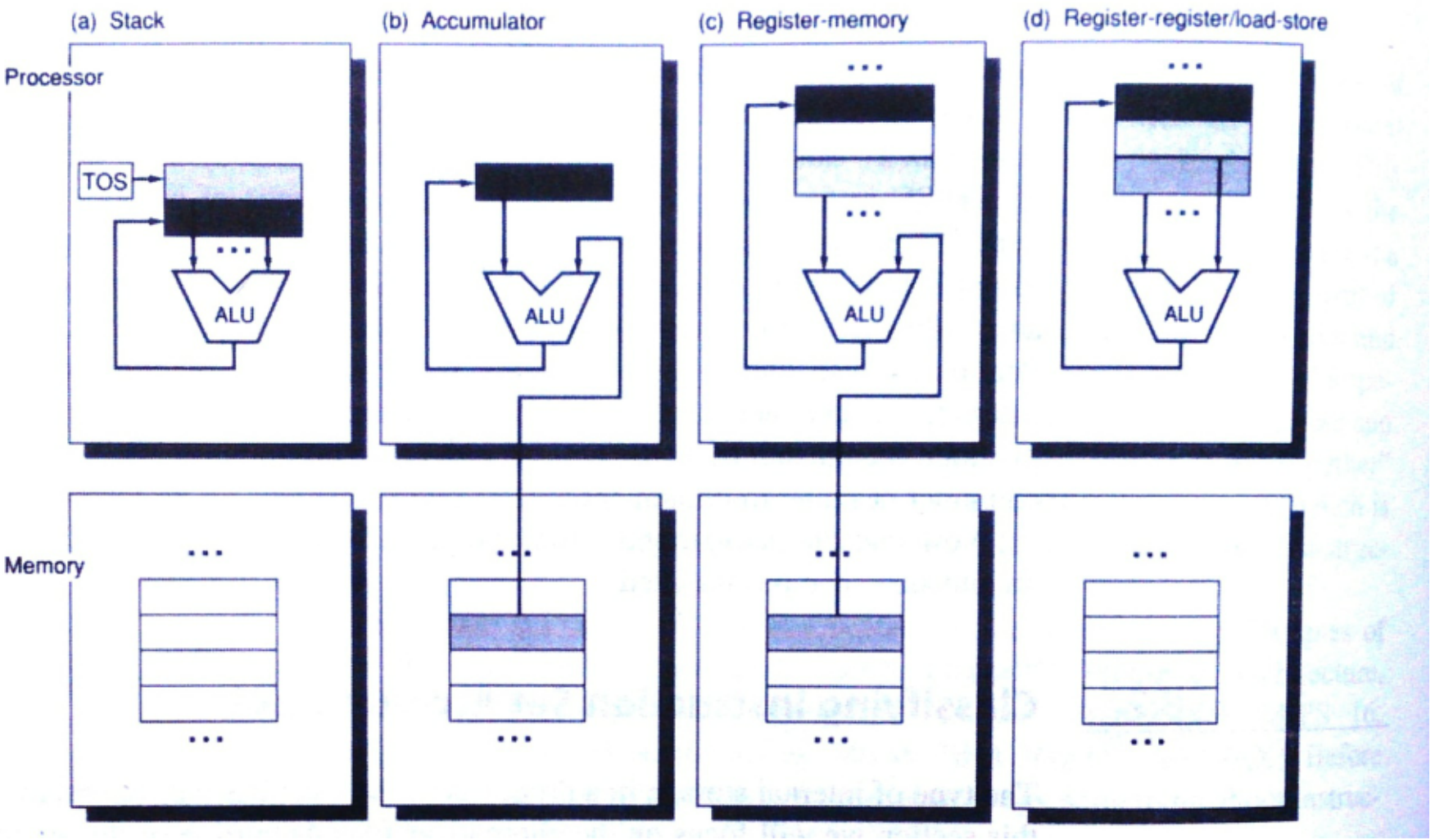
- Storage Components
  - Registers
  - Flags
- Execution (Processing) Components
  - Arithmetic Logic Unit(ALU)
    - Arithmetic calculations, Logical computations, Shifts/Rotates
- Transfer Components
  - Bus
- Control Components
  - Control Unit



PROCESSOR ORGANIZATION

- In general, most processors are organized in one of 3 ways
  - Stack organization
    - » All operations are done using the hardware stack
    - » For example, an OR instruction will pop the two top elements from the stack, do a logical OR on them, and push the result on the stack
  - Single register (Accumulator) organization
    - » Basic Computer is a good example
    - » Accumulator is the only general purpose register
  - General register organization
    - » Used by most modern computer processors
    - » Any of the registers can be used as the source or destination for computer operations

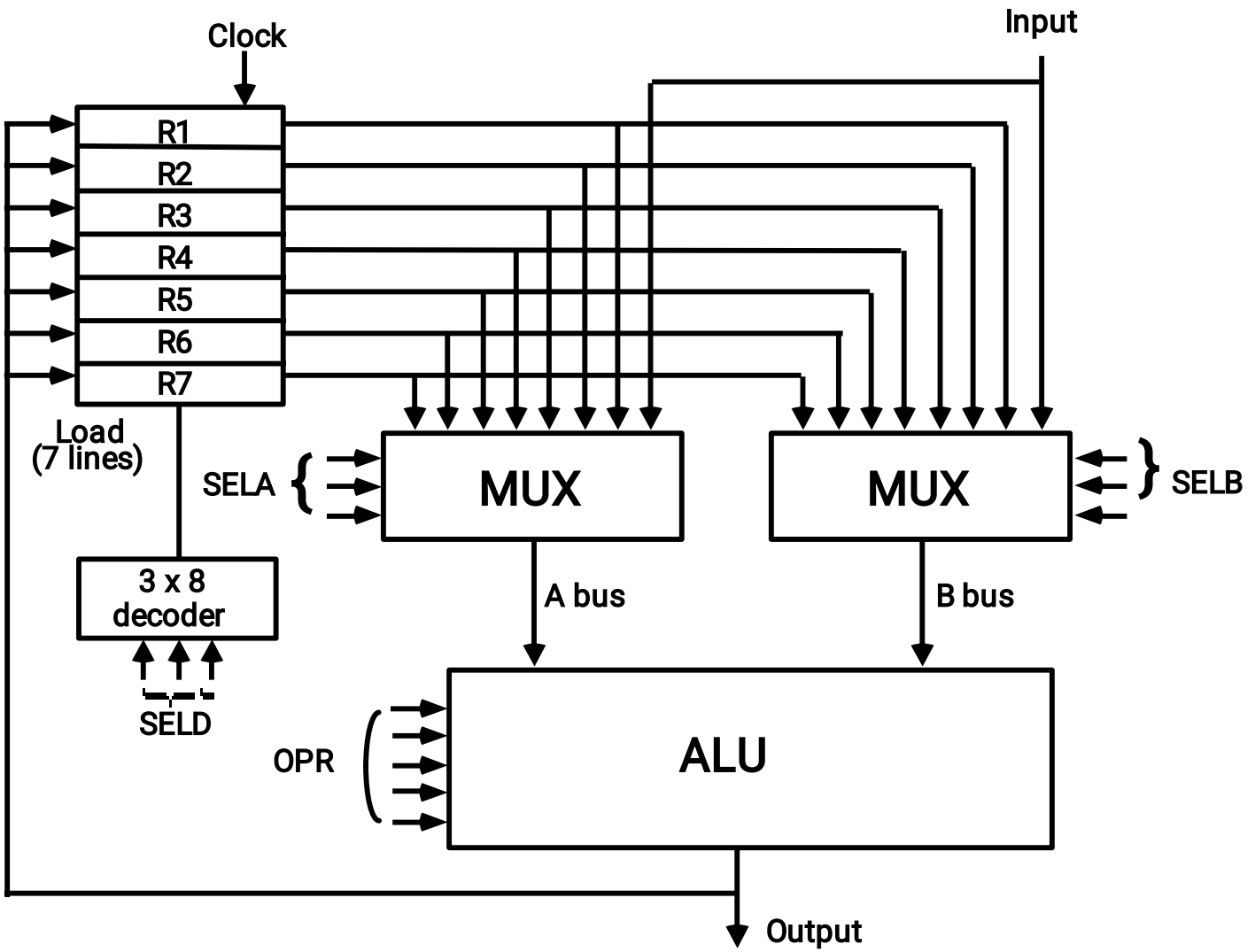
Stack, accumulator and general purpose register organization.



Code Sequence Example for C=A+B

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

GENERAL REGISTER ORGANIZATION



OPERATION OF CONTROL UNIT

The control unit

Directs the information flow through ALU by

- Selecting various *Components* in the system
- Selecting the *Function* of ALU

Example: R1 ← R2 + R3

[1] MUX A selector (SELA): BUS A ← R2

[2] MUX B selector (SELB): BUS B ← R3

[3] ALU operation selector (OPR): ALU to ADD

[4] Decoder destination selector (SELD): R1 ← Out Bus

Control Word

Encoding of register selection fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

ALU CONTROL

Encoding of ALU operations

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	ADD A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Examples of ALU Microoperations

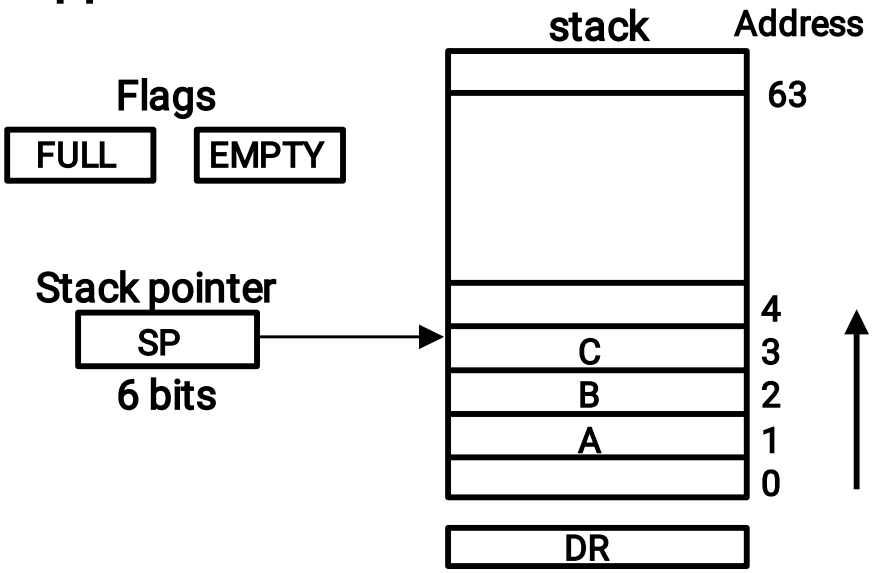
Microoperation	Symbolic Designation SELA	SELB	SELD	OPR	Control Word
R1 ← R2 – R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 ∨ R5	R4	R5	R4	OR	100 101 100 01010
R6 ← R6 + 1	R6	-	R6	INCA	110 000 110 00001
R7 ← R1	R1	-	R7	TSFA	001 000 111 00000
Output ← R2	R2	-	None	TSFA	010 000 000 00000
Output ← Input	Input	-	None	TSFA	000 000 000 00000
R4 ← shl R4	R4	-	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

REGISTER STACK ORGANIZATION

Stack

- Very useful feature for nested subroutines, nested interrupt services
- Also efficient for arithmetic expression evaluation
- Storage which can be accessed in LIFO
- Pointer: SP
- Only PUSH and POP operations are applicable

Register Stack



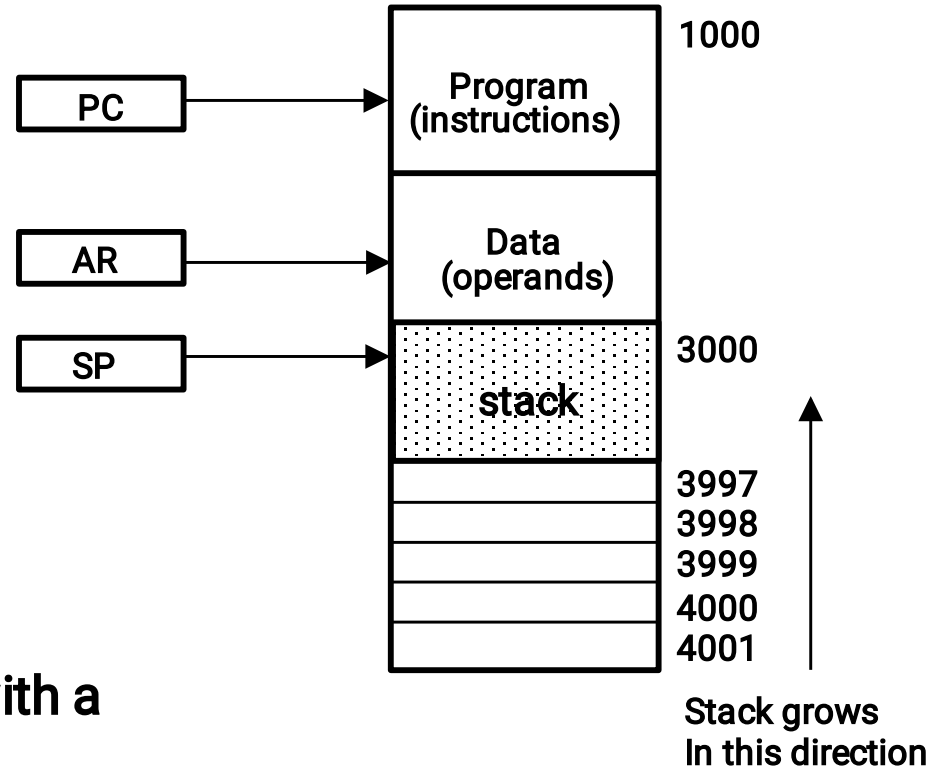
Push, Pop operations

/* Initially, SP = 0, EMPTY = 1, FULL = 0 */

PUSH	POP
SP ← SP + 1	DR ← M[SP]
M[SP] ← DR	SP ← SP – 1
If (SP = 0) then (FULL ← 1)	If (SP = 0) then (EMPTY ← 1)
EMPTY ← 0	FULL ← 0

MEMORY STACK ORGANIZATION

Memory with Program, Data, and Stack Segments



- A portion of memory is used as a stack with a processor register as a stack pointer

- PUSH: SP ← SP - 1
  M[SP] ← DR
- POP: DR ← M[SP]
  SP ← SP + 1

- Most computers do not provide hardware to check stack overflow (full stack) or underflow (empty stack) → must be done in software

REVERSE POLISH NOTATION

- Arithmetic Expressions: A + B

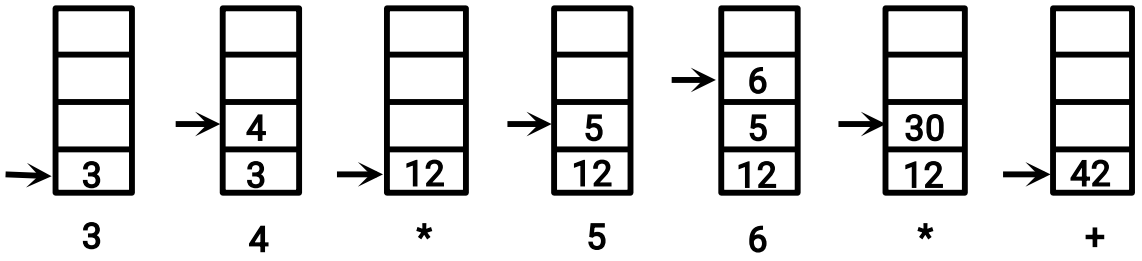
  A + B Infix notation
  + A B Prefix or Polish notation
  A B + Postfix or reverse Polish notation

  - The reverse Polish notation is very suitable for stack manipulation

- Evaluation of Arithmetic Expressions

  Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation

(3 * 4) + (5 * 6) ⇒ 3 4 * 5 6 * +



INSTRUCTION FORMAT

- Instruction Fields

  OP-code field - specifies the operation to be performed

  Address field - designates memory address(es) or a processor register(s)

  Mode field - determines how the address field is to be interpreted (to get effective address or the operand)

- The number of address fields in the instruction format depends on the internal organization of CPU

- The three most common CPU organizations:

  Stack organization:

  ```
  PUSH X          /* TOS ← M[X]  */
  ADD
  ```

  Single accumulator organization:

  ```
  ADD X           /* AC ← AC + M[X]  */
  ```

  General register organization:

  ```
  ADD R1, R2, R3  /* R1 ← R2 + R3  */
  ADD R1, R2      /* R1 ← R1 + R2  */
  ```

THREE, AND TWO-ADDRESS INSTRUCTIONS

- Three-Address Instructions

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      */
```

ONE, AND ZERO-ADDRESS INSTRUCTIONS

- 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
  - 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  */
```