CPU

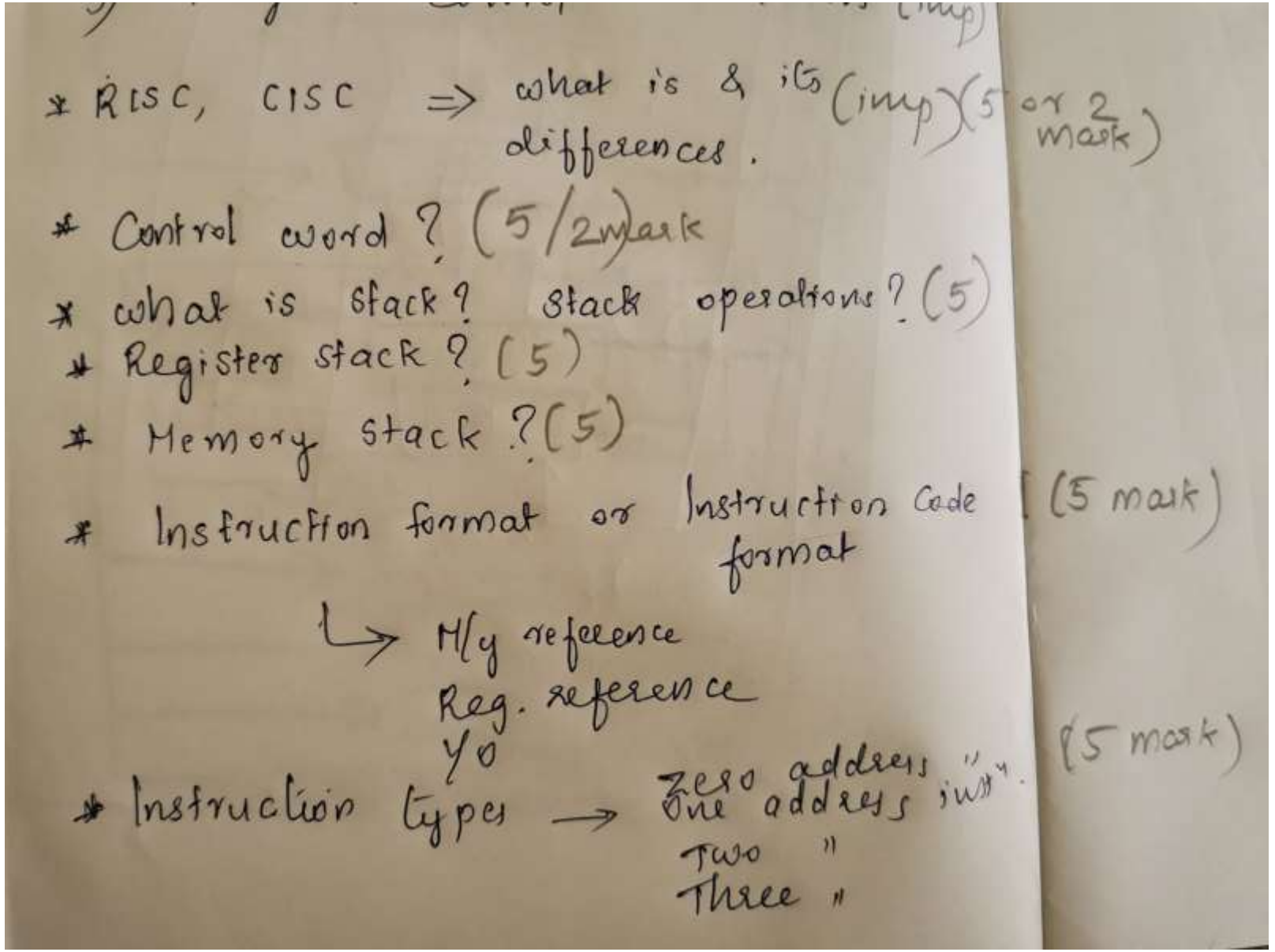
Unit 2

CPU

Topics (chap. 8 → 8-2, 8-3, 8-4, 8-5, 8-6, 8-7)

1) General Reg. organization (imp)
2) stack organization (imp)
3) Addressing Modes (imp)
4) Instruction classification
5) Program Control Instructions (imp)

RISC, CISC ⇒ what is & its differences. (imp)


* RISC, CISC ⇒ what is & its differences. (imp)(5 or 2 mark)
* Control word ? (5/2 mark
* what is stack ? stack operations ? (5)
* Register stack ? (5)
* Memory stack ? (5)
* Instruction format or Instruction Code format (5 mark)
↳ M/y reference
Reg. reference
Yo
* Instruction types → Zero address ins".
One address ins".
Two "
Three "
(5 mark)

Instruction classifications

- Data transfer (5)
- Data manipulation (5)
- Pgm Control (5 or essay)

Addressing modes (essay / 5 mark)

- Implied
  - Immediate
  - Reg.
  - Reg. indirect
  - Direct
  - Indirect
  - Auto incre/Auto decre
  - Relative
  - Indexed
  - Base Reg.

CPU

- The part of the computer that performs the bulk of data-processing operations
- 3 parts:
  - Registers
  - ALU
  - Control unit

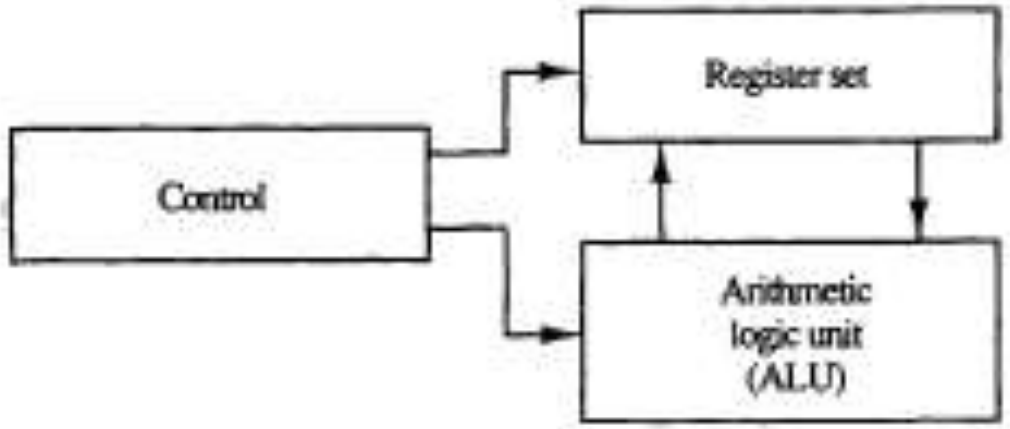
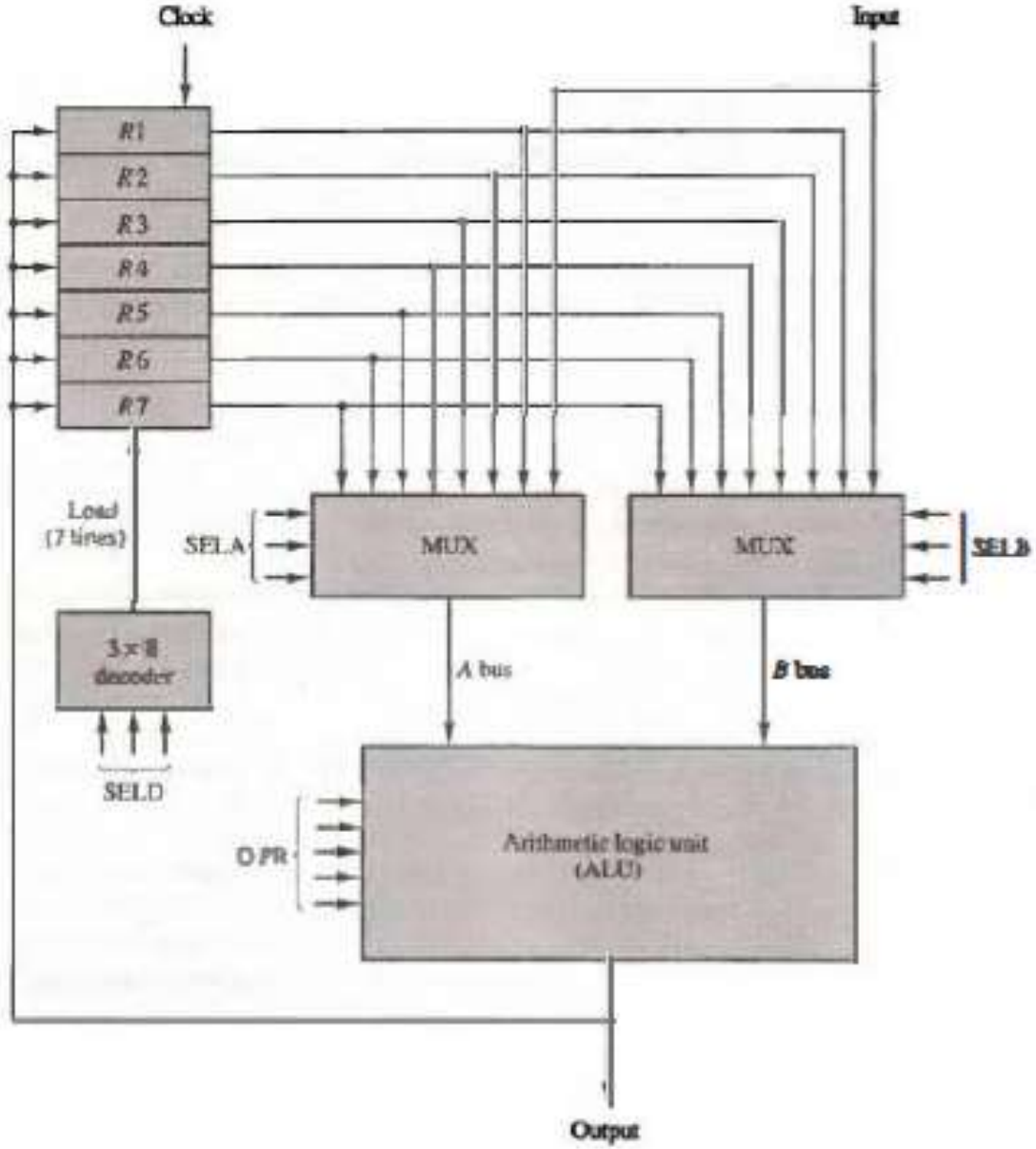


Figure 8-1 Major components of CPU.

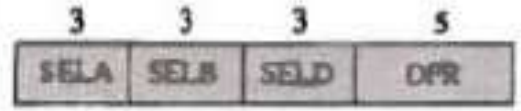
- The register set stores intermediate data used during the execution of the instructions
- The arithmetic logic unit (ALU) performs the required micro-operations for executing the instructions.
- The control unit supervises the transfer of information among the registers and instructs the ALU as to which operation to perform.

General Register Organization

- Memory locations are needed for storing pointers, counters, return addresses, temporary results, and partial products during multiplication.
- Time consuming
- It is more convenient and more efficient to store these intermediate values in processor registers.
- When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system.
- The registers communicate with each other not only for direct data transfers, but also while performing various micro-operations.



(a) Block diagram



(b) Control word

Figure 8-2 Register set with common ALU.

- The output of each register is connected to two multiplexers (MUX) to form the two buses A and B .

- The selection lines in each multiplexer select one register or the input data for the particular bus.

- The A and B buses form the inputs to a common arithmetic logic unit (ALU). The operation selected in the ALU determines the arithmetic or logic micro-operation that is to be performed.

- The result of the micro-operation is available for output data and also goes into the inputs of all the registers.

- The register that receives the information from the output bus is selected by a decoder.

- The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register.

- The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system.

- For example, to perform the operation R 1 ← R2 + R3

- The control must provide binary selection variables to the following selector inputs

- 1. MUX A selector (SELA): to place the content of R2 into bus A .

- 2 . MUX B selector (SELB): to place the content o f R 3 into bus B .

- 3 . ALU operation selector (OPR): to provide the arithmetic addition A + B .

- 4. Decoder destination selector (SELD): t o transfer the content o f the output bus into R 1 .

- The four control selection variables are generated in the control unit

- The data from the two source registers propagate through the gates in the multiplexers and the ALU, to the output bus, and into the inputs of the destination register, all during the clock cycle interval.

- when the next clock transition occurs, the binary information from the output bus is transferred into R 1.

Control Word

- There are 14 binary selection inputs in the unit, and their combined value specifies a control word (14 bit)
- The three bits of SELA select a source register for the A input of the ALU.
- The three bits of SELB select a register for the B input of the ALU.
  of the ALU
- The three bits of SELD select a destination register using the decoder and its seven load outputs.
- The five bits of OPR select one of the operations in the ALU.
  ALU
- The 14-bit control word when applied to the selection inputs specify a particular micro-operation

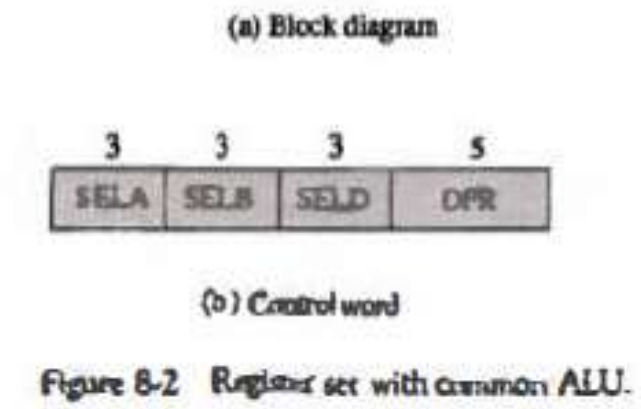


Figure 8-2 Register set with common ALU.

TABLE 8-1 Encoding of Register Selection Fields

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	*R*1	*R*1	*R*1
010	*R*2	*R*2	*R*2
011	*R*3	*R*3	*R*3
100	*R*4	*R*4	*R*4
101	*R*5	*R*5	*R*5
110	*R*6	*R*6	*R*6
111	*R*7	*R*7	*R*7

STACK ORGANIZATION

STACK

- A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved
- a memory unit with an address register that can count only (after an initial value is loaded into it).
- The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top item in the stack
- the physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted
- The operation of insertion is called push (or push-down) because it can be thought of as the result of pushing a new item on top.
- The operation of deletion is called pop (or pop-up) because it can be thought of as the result of removing one item so that the stack pops up
- Incrementing or decrementing the stack pointer register

Register Stack

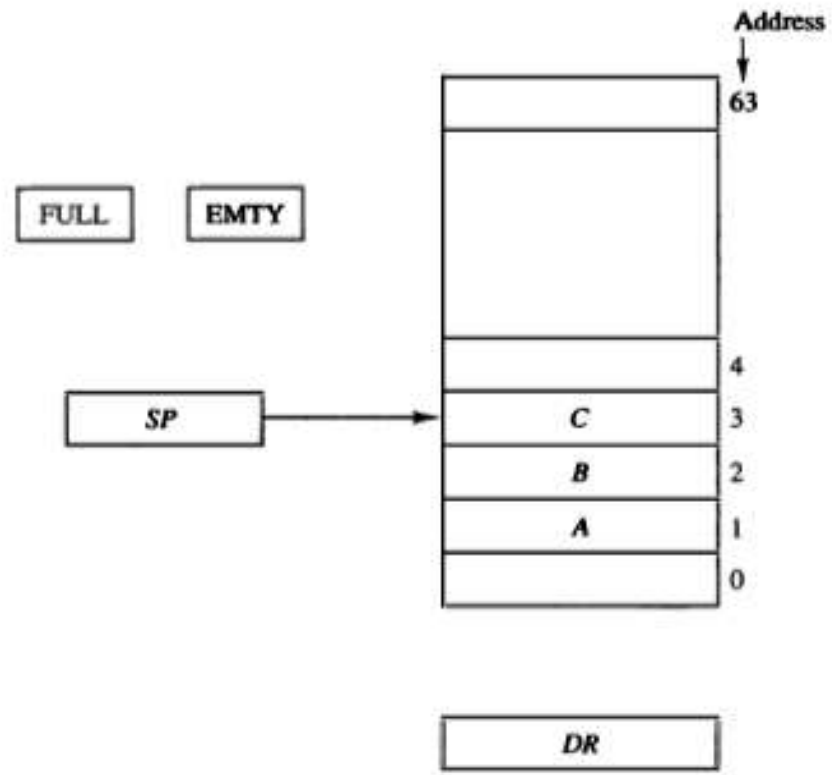


Figure 8-3 Block diagram of a 64-word stack.

- 64-word register stack.
- The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack.
- Three items are placed in the stack: A, B, and C, in that order.
- Item C is on top of the stack so that the content of SP is now 3.
- To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP
- To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack

eg:

- In a 64-word stack, the stack pointer contains 6 bits because $2^6 = 64$.
- Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary).
- When63isincrementedby1,th eresult is 0, since111111 + 1 = 1000000 in binary, but SP can accommodate only the six least significant bits.
- Similarly, when 000000 is decremented by 1, the result is 111111.
- The one-bit register FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items
- DR is the data register that holds the binary data to be written into or read out of the stack

push operation

- Initially, SP is cleared to 0, EMTY is set to 1, and FULL is cleared to 0.
- If the stack is not full (if FULL = 0), a new item is inserted with a push operation.

- The push operation is implemented with the following sequence of micro-operations
  - SP ←SP + 1 — Increment stack pointer
  - M[SP]←DR — Write item on top of the stack

  - If (SP = 0) then (FULL ←1) — Check if stack is full
  - EMTY ←0 — Mark the stack not empty

- The stack pointer is incremented so that it points to the address of the next-higher word.
- A memory write operation inserts the word from DR into the top of the stack

- SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP.
- The first item stored in the stack is at address 1
- The last item is stored at address 0.
- If SP reaches 0, the stack is full of items, so FULL is set to 1
- This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0.
- Once an item is stored in location 0, there are no more empty registers in the stack.

The pop operation

- A new item is deleted from the stack if the stack is not empty (if EMTY = 0).

DR ←M[SP]	Read item from the top of stack
SP←SP – 1	Decrement stack pointer
If (SP = 0) then (EMTY ←1)	Check if stack is empty
FULL ←0	Mark the stack not full

- The top item is read from the stack into DR.
- The stack pointer is then decremented.
- If its value reaches zero, the stack is empty, so EMTY is set to 1
- This condition is reached if the item read was in location 1

- Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP.

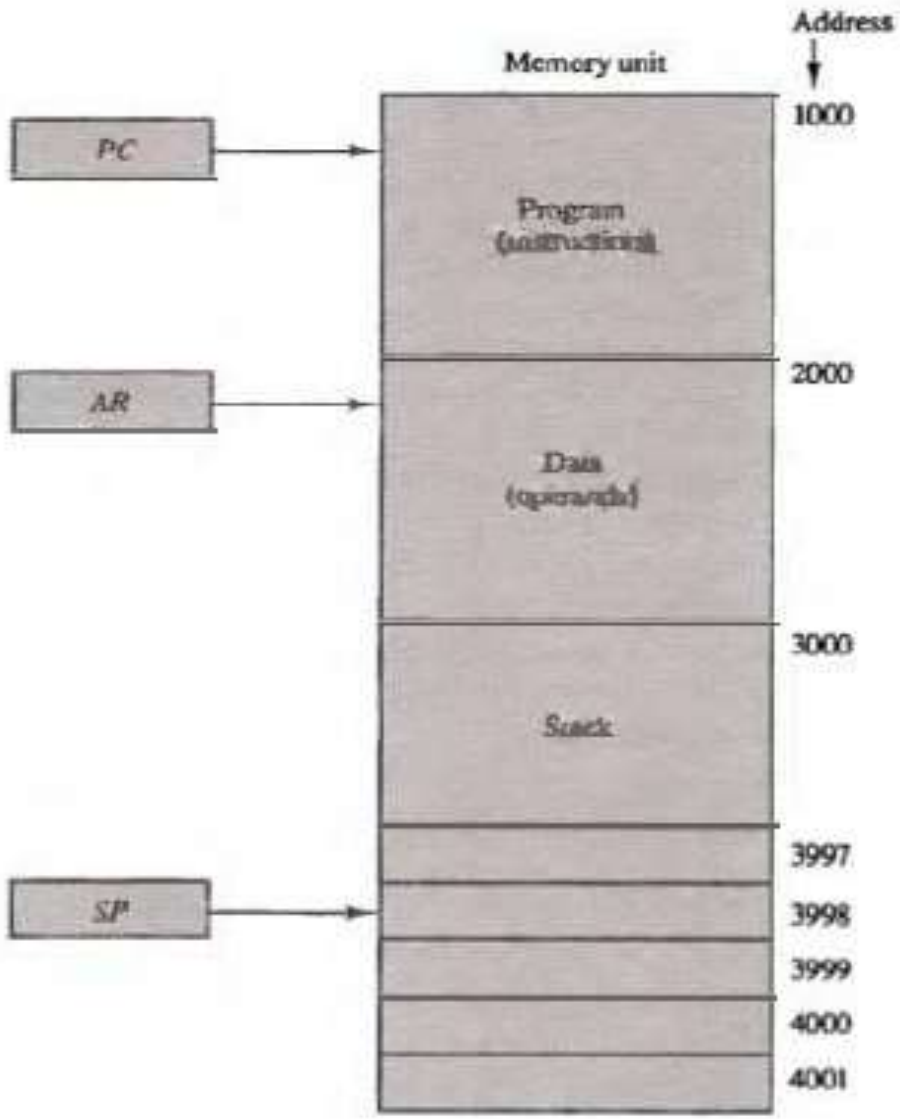
- if a pop operation reads the item from location 0 and then SP is decremented, SP changes to 111111, which is equivalent to decimal 63.

- In this configuration, the word in address 0 receives the last item in the stack.

- Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1

Memory Stack

- A stack can exist as a stand-alone unit or can be implemented in a random-access memory attached to a CPU.
- The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and
- using a processor register as a stack pointer.
- a portion of computer memory partitioned into three segments: program, data, and stack.
- The program counter PC points at the address of the next instruction in the program.
- The address register AR points at an array of data.


Address
Memory unit
PC
AR
SP
Program
(instructions)
Data
(operands)
Stack
1000
2000
3000
3997
3998
3999
4000
4001

- SP points at the top of the stack.
- The three registers are connected to a common address bus, and either one can provide an address for memory.
- PC is used during the fetch phase to read an instruction.
- AR is used during the execute phase to read an operand.
- SP is used to push or pop items into or & from the stack.
- the initial value of SP is 4001 and the stack grows with decreasing addresses.
- Thus the first item stored in the stack is at address 4000 , the second item is stored at address 3999, and the last address that can be used for the stack Is 3000.

- A new item is inserted with the push operation as follows:
  - SP ←SP - 1
  - M[SP] ←DR

- The stack pointer is decremented so that it points at the address of the next word.

- A memory write operation inserts the word from DR into the top of the stack.

- A new item is deleted with a pop operation as follows:
  - DR ←M[SP]
  - SP ←SP + 1

- The top item is read from the stack into DR .

- The stack pointer is then incremented to point at the next item in the stack.

Reverse Polish Notation

- A stack organization is very effective for evaluating arithmetic expressions.
- The common arithmetic expressions are written in infix notation, with each operator written between the operands.

  A * B + C * D

- To evaluate arithmetic expressions in infix notation it is necessary to scan back and forth along the expression to determine the next operation to be performed.

- prefix notation . This representation, often referred to as Polish notation, places the operator before the operands

- The postfix notation, referred to as reverse Polish notation (RPN), places the operator after the operands.

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

- The reverse Polish notation is in a form suitable for stack manipulation. The expression
    - A * B + C * D
- is written in reverse Polish notation as
    - AB * CD * +

- Scan the expression from left to right.
- When an operator is reached, perform the operation with the two operands found on the left side of the operator.
- Remove the two operands and the operator and replace them by the number obtained from the result of the operation
- Continue to scan the expression and repeat the procedure for every operator encountered until there are no more operators

- For the expression above we find the operator * after A and B.

- We perform the operation A * B and replace A, B, and * by the product to obtain

- (A * B) CD * +
- where (A * B) is a single quantity obtained from the product.

- The next operator is a * and its previous two operands are C and D, so we perform C * D and obtain an expression with two operands and one operator:
- (A * B) (C * D) +

- The next operator is + and the two operands to be added are the two products, so we add the two quantities to obtain the result.

Eg:2

- we first perform all arithmetic inside inner parentheses, then inside outer parentheses, and do multiplication and division operations before addition and subtraction operations.

- Consider the expression

  (A + B) • [C • (D + E) + f]

  parentheses (A + B) and (D + E).
- inside the square brackets.

- The multiplication of C • (D + E) must be done prior to the addition of F since multiplication has precedence over addition.

- The last operation is the multiplication of the two terms between the parentheses and brackets.

(A + B) * [C * (D + E) + F]

AB + DE + C * F + *

(A + B)(D + E)C * F + *

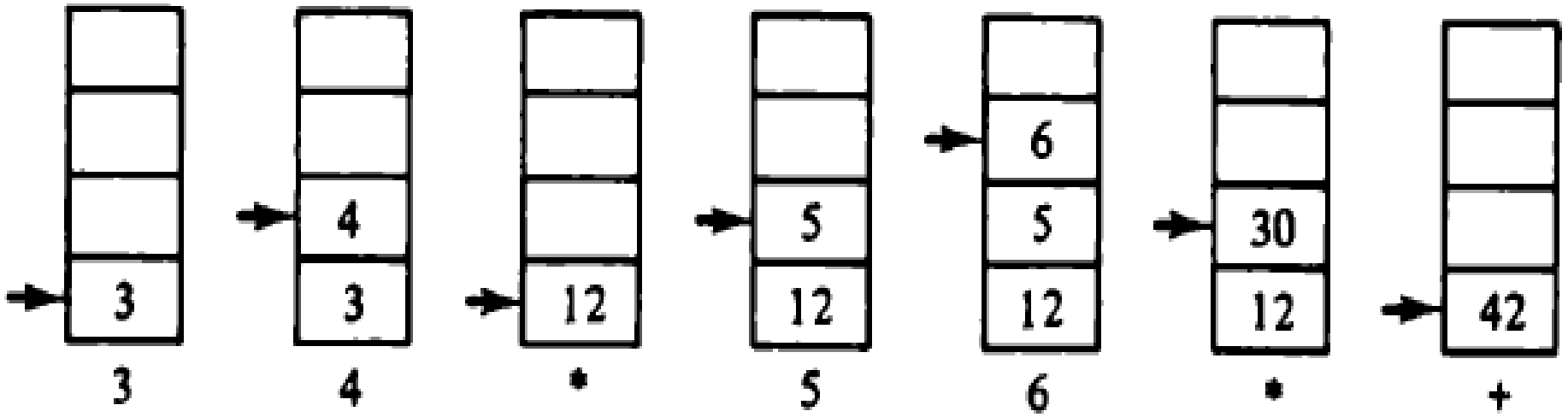
Evaluation of Arithmetic Expressions

(3 * 4) + (5 * 6)

- In reverse Polish notation, it is expressed as

  34 * 56 * +

Figure 8-5 Stack operations to evaluate 3 • 4 + 5 • 6.



Instruction Formats

Instruction Formats

- 1. An operation code field that specifies the operation to be performed.
- 2. An address field that designates a memory address or a processor register.
- 3. A mode field that specifies the way the operand or the effective address is determined.
- Operands residing in memory are specified by their memory address.
- Operands residing in processor registers are specified with a register address.
- A register address is a binary number of k bits that defines one of 2^K registers in the CPU.

- CPU organizations:
    1. Single accumulator organization.
    2. General register organization.
    3. Stack organization

1. Single CPU

A D D X

where X is the address of the operand.

The ADD instruction in this case result in the operation AC <--AC + M [X].

AC is the accumulator register and M [X]symbolizes the memory word located at address X.

- A general register type of organization

  A D D R 1 , R 2 , R 3 to denote the operation R 1 <--- R2 + R 3 .

The number o f address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers.

- Thus the instruction

  A D D R 1 , R 2

  would denote the operation R 1 <--- R 1 + R2. Only register addresses for R 1 and R2 need be specified in this instruction.

- A D D R 1 , X would specify the operation Rl <--R l + M [X]. It has two address fields, one for register R 1 and the other for the memory address X .

- stack-organized CPU

  P U S H X will push the word at address X to the top of the stack

A. Three-Address Instructions

X = (A + B) * (C + D)

A DD R 1 , A , B	R 1 ←M [A] + M [B]
A DD R 2 , C , D	R 2 ←M [C] + M [D]
M U L X , R 1 , R 2	M [X] ←R 1 *R 2

B. Two-Address Instructions

X = (A + B) * (C + D)

M O V R 1 , A	R 1 <--M [A]
A D D R 1 , B	R 1 <--R 1 + M [B]
M O V R 2 , C	R 2 <--M [C]
A D D R 2 , D	R 2 <--R 2 + M [D]
M U L R 1 , R 2	R 1 <--R 1 * R 2
M O V X , R 1	M [X] <--R 1

C. One-Address Instructions

use an implied accumulator (AC) register for all data manipulation

X = (A + B) * (C + D)

L O A D A	A C <- M [A]
A DD B	A C <- A C + M [B]
S TORE T	M [T] <- A C
LOAD C	A C <- M [C]
ADD D	A C <- A C + M [D]
MUL T	A C <- A C * M [T]
STORE X	M [X] <- A C

D. Zero-Address Instructions

- stack-organized computer does not use an address field for the instructions ADD and MUL

X = (A + B) * (C + D)

PUSH A	T O S <- A
PUSH B	T O S <- B
ADD	T O S <- (A + B)
PUSH C	T O S <- C
PUSH D	T O S <- D
ADD	T O S <- (C + D)
MUL	T O S <- (C + D) * (A + B)
POP X	M [X] <- T O S

E. RISC Instructions

add and multiply operations are executed with data in the register without accessing memory. The result of the computations is then stored I memory with a store instruction.

- reduced instruction set computer (RISC) architecture
- use of load and store instructions

```
LOAD   R 1 , A          R 1 <--M [ A ]
LOAD   R 2 , B          R 2 <--M [ B ]
LOAD   R3 , C           R3 <--M [ C ]
LOAD   R  , D           RL; <--M [ D ]
ADD    R 1 , R 1 , R 2  R 1 <--R 1 + R 2
ADD    R 3 , R 3 , R 2  R 3 <--R 3 + R L;
MUL   R 1 , R 1 , R 3   R 1 <--R 1 * R 3
STORE X , R 1           M [ X ] <--R 1
```

INSTRUCTION CLASSIFICATION

1. Data transfer instructions
2. Data manipulation instructions
3. Program control instructions

- Data transfer instructions cause transfer of data from one location to another without changing the binary information content.
-
- Data manipulation instructions are those that perform arithmetic, logic, and shift operations.

- Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer.

- The instruction set of a particular computer determines the register transfer operations and control decisions that are available to the user.

1. Data Transfer Instructions

- Data transfer instructions move data from one place in the computer to another without changing the data content.

TABLE 8-5 Typical Data Transfer Instructions

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Manipulation Instructions

- perform operations on data and provide the computational capabilities for the computer.

- usually divided into three basic types:

    1. Arithmetic instructions
    2. Logical and bit manipulation instructions
    3. Shift instructions

Arithmetic Instructions: addition, subtraction, multiplication, and division.

TABLE 8-7 Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

ADD I	Add two binary integer numbers
ADD F	Add two floating - point numbers
ADD D	Add two decimal numbers in B C D

Logical and Bit Manipulation Instructions

- perform binary operations on strings of bits stored in register
- They are useful for manipulating individual bits or a group of bits that represent binary-coded information

TABLE 8-8 Typical Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

- The AND instruction is used to clear a bit or a selected group of bits of an operand.
- For any Boolean variable x, the relationships xb0 = 0 and x b1= x

TABLE 8-8 Typical Logical and Bit Manipulation Instructions

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

- The OR instruction is used to set a bit or a selected group of bits of an operand.

- For any Boolean variable x, the relationships x + 1 = 1 and x + 0 = x dictate that a binary variable OR ed with a 1 produces a 1; but the variable does not change when OR ed with a 0.

- Therefore, the OR instruction can be used to selectively set bits of an operand by OR ing it with another operand with 1' s in the bit positions that must be set to 1 .

- XOR instruction i s used t o selectively complement bits of bits an operand.

Shift Instructions

- The bit shifted in at the end of the word determines the type of shift used.

- Shift instructions may specify either logical shifts, arithmetic shifts, or rotate-type operations.

TABLE 8-9 Typical Shift Instructions

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

- The logical shift inserts 0 to the end bit position.
- The end position is the leftmost bit for shift right and the rightmost bit position for the shift left.
- Arithmetic shifts usually conform with the rules for signed-2' s complement numbers
- The rotate instructions produce a circular shift

ADDRESSING MODES

Addressing Modes

1. Implied Mode
2. Immediate Mode
3. Register Mode
4. Register Indirect Mode
5. Auto-increment or Auto-decrement Mode
6. Direct Address Mode
7. Indirect Address Mode
8. Relative Address Mode
9. Indexed Addressing Mode
10. Base Register Addressing Mode

Implied Mode

- In this mode the operands are specified implicitly in the definition of the instruction.
- For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction.
- all register reference instructions that use an accumulator are implied
- Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack. ode instructions

Immediate Mode

- In this mode the operand is specified in the instruction itself.
- an immediate-mode instruction has an operand field rather than an address field.
- The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.
- Immediate-mode instructions are useful for initializing registers to a constant value.

Register Mode

- In this mode the operands are in registers that reside within the CPU.
- The particular register is selected from a register field in the instruction.
- A k-bit field can specify any one of 2^k registers

Register Indirect Mode

- In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.
- The selected register contains the address of the operand rather than the operand itself
- The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

Auto-increment or Auto-decrement Mode

- similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory.
- When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.
- This can be achieved by using the increment or decrement instruction

Direct Address Mode

- In this mode the effective address is equal to the address part of the instruction.
- The operand resides in memory and its address is given directly by the address field of the instruction

Indirect Address Mode

- In this mode the address field of the instruction gives the address where the effective address is stored in memory.

- Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

- effective address = address part of instruction + content of CPU register
  (an index register, or a base register)

Relative Address Mode

- In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address.
- The address part of the instruction is usually a signed number (in 2' s complement representation) which can be either positive or negative.
- When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction.
- Eg: the program counter contains the number 825 and the address part of the instruction contains the number 24.
- The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to 826.
- The effective address computation for the relative address mode is 826 + 24 = 850. This is 24 memory locations forward from the address of the next instruction.
- used with branch-type instructions

Indexed Addressing Mode

- The content of an index register is added to the address part of the instruction to obtain the effective address.
- The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory.
- Each operand in the array is stored in memory relative to the beginning address.
- The distance between the beginning address and the address of the operand is the index value stored in the index register.
- Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value.
- The index register can be incremented to facilitate access to consecutive operands
- does not include an address field in its format

Base Register Addressing Mode

- The content of a base register is added to the address part of the instruction to obtain the effective address.
- The difference between the two modes is in the way they are used rather than in the way that they are computed.

- An index register is assumed to hold an index number that is relative to the address part of the instruction.

- A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address.

- The base register addressing mode is used in computers to facilitate the relocation of programs in memory.

TABLE 8-6 Eight Addressing Modes for the Load Instruction

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD $ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$