CPU ORGANIZATION

The part of computer that do data processing operations is called central processing unit (CPU)
The CPU is made of 3 parts:

1. **Registers**: stores intermediate data generated during execution
2. **ALU**: performs required micro operations
3. **Control Unit**: controls transfer of data among registers and instruct ALU to perform correct operation

General Register Organization

Intermediate data are needed to be stored like pointers, counters, return address, temp results, and partial products.

Cannot save them in main memory because their access is time consuming.

It is more efficient and faster to be stored inside processor.

So the solution is designing multiple registers inside processor and connects them through a common bus.

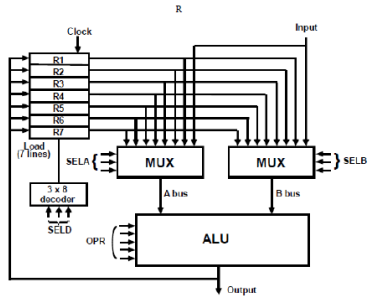
In Basic Computer, there is only one general purpose register, the Accumulator (AC) but in modern CPUs, there are many general purpose registers. It is advantageous to have many registers
Transfer between registers within the processor are relatively fast. Going "off the processor" to access memory is much slower.

BUS SYSTEM

A new bus organization will be introduced in order to clarify the idea of register banking and how to control their actions.
7 CPU registers that their outputs are connected to 2 MUX 8 X 1 to form the 2 buses A and B.

The A and B are inputted to ALU unit in which its operation is selected by their select lines among different arithmetic and logic operations.

The resulted ALU data can is directed to the input of all 7 registers which one of them will be selected according to 3 X 8 decoder connected to LD inputs of the registers



For example to perform operation
R1 ←R2 + R3
MUXA select R2 , MUXB select R3 , OPR in ALU operation for ADD , SELD to direct destination register R1.

These four control signals are generated in control unit in start of each clock cycle ensuring operands are selected beside correct ALU operation and result is chosen in one clock cycle only.

CONTROL WORD

There are 14 selection inputs in the unit and their combined value specifies control word.

3	3	3	5
SELA	SELB	SELD	OPR

bits to select A source, 3 bits to select B source, 5 bits to select operation required on them, and finally 3 bits to select destination register.

Encoding of 3 bits for selection of the 2 sources plus the destination is defined in next table. While the other table specifies ALU operations encoding.

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

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

ALU

ALU provides arithmetic operations (ADD, SUB, INCA, DECA)

Logic operations (AND, OR, XOR, COMA)

Shift operations (SHLA SHRA).

And Transfer operation (TSFA)

EXAMPLES

1. Derive a control word that executes the next statement

R1 ←R2 – R3

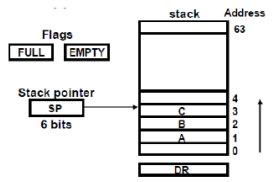
Field	SELA	SELB	SELD	OPR
Symbol	R2	R3	R1	SUB
CW	010	011	001	00101

Stack Organization

Stack is a storage device that stores information in a way that the item is stored last is the first to be retrieved (LIFO).
Stack in computers is actually a memory unit with address register (stack pointer SP) that can count only. SP value always points at top item in stack.
The two operations done on stack are
PUSH (Push Down), operation of insertion of items into stack
POP (Pop Up), operation of deletion item from stack
Those operation are simulated by INC and DEC stack register (SP).

1**. Register stack**

A stand alone unit that consists of collection of finite number of registers.
64 location stack unit with SP that stores address of the word that is currently on the top of stack.



3 items are placed in the stack A, B, and C. Item C is in top of stack so that SP holds 3 which the address of item C.

To remove top item from stack (popping stack) we start by reading content of address 3 and decrementing the content of SP. Item B is now in top of stack holding address 2.
To insert new item (pushing the stack) we start by incrementing SP then writing a new word where SP now points to (top of stack).

in 64 word stack we need to have SP of 6 bits only (from 000000 to 111111). If 111111 is reached then at next push SP will be 000000, that is when the stack is FULL. Similarly when SP is 000001 then at next pop SP will go to 000000 that is when the stack is EMTY.
Initially, SP = 0, EMPTY = 1, FULL =0
Procedures for pushing stack
SP ←SP + 1
M[SP] ←DR
IF (SP = 0) THEN (FULL = 1)
EMTY ←0

1. Always we use DR to pass word into stack
2. M[SP] memory word specified by address currently in SP
3. First item stored in stack is at address 1
4. Last item stored in stack is at address 0. That is FULL = 1
5. Any push to stack means EMTY = 0

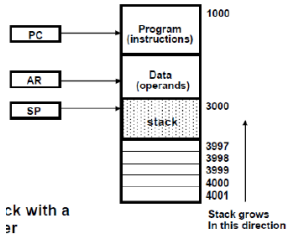
Procedures for popping stack
DR ←M[SP]
SP ←SP – 1
IF (SP = 0) THEN (EMTY = 1)
FULL ←0
Note:
1. Top of stack is read into DR
2. If SP reached 0 then stack is EMTY = 1. That when SP was 1 then pop occurred. No more pops can happen from here.
3. Any pop from stacks means FULL = 0

Memory Stack :

Stack can be implemented in RAM memory attached to CPU. Only by assigning special part of it for stack operations.
Next figure shows of main memory divided into program, data, and stack.
o PC points to next instruction in instruction part
o AR points to array of data of operands
o SP points to top of stack
o All are connected to common address bus
Stack grows (pushed) with decreasing address and empties (pops) with increasing address.
New item is inserted with push operation by decrementing SP then a write to SP address is done
SP ←SP -1
M [SP] ←DR

Last item is removed from stack with pop operation by removing item by reading from memory location addressed by SP then SP is incremented.

DR ←M [SP]
SP ←SP + 1



initial value of SP is 4001 and first item when pushed in stack stores at address 4000 and second one stores at address 3999. The last address pushed into will be 3000. (See limitation danger?)

Most computers are not supported by hardware to sense stack overflow and underflow. But can be implemented by saving the 2 limits in 2 registers. After each push or pop the SP is compared with the limit to see if stack has reached its limits. So must be taking care of using software.

3. Reverse Polish Notation

Very useful notation to utilize stacks to evaluate arithmetic expressions.

infix notation:
A*B + C*D

We compute A*B, store product, compute C*D, then sum two products. So we have to scam back and forth to see which operation comes first.

The 3 notations to evaluate expressions
1. A + B Infix notation
2. +AB Prefix notation (Polish notation)
3. AB+ Postfix notation (reverse Polish notation)

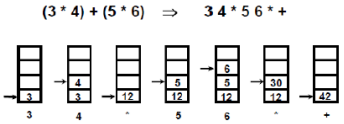
Reverse Polish Notation is in a form suitable for stack manipulation. Starts by scanning expression from left to right. When operator is found then perform operation with 2 operands in left of operator and replace result place of 2 operands and operator. Then we can continue this until you reach final answer.

Example
Expression A*B + C*D is written in RPN as AB*CD*+. And will be computed as
(A*B) CD *+
(A*B)(C*D)+

Example
Convert infix notation expression (A + B)*(C * (D + E) + F) to RPN?
AB+ DE+ C * F+ *. Will be computed as (A+B) (D+E) C * F + *

Reverse polish notation combined with stack comprised of registers is most efficient way to evaluate expression. Stacks are good for handling long and complex problems involving chain calculations. But need first to convert arithmetic expressions into parenthesis-free reverse polish notation.

This procedure is employed in some scientific calculators and some computers



Instruction Format

The Instruction coding fields in today's computers follow the next format
1. Operation code field to specify operation
2. Address field that specifies operand address field or register
3. Mode field to specify effective address

In general, most processors are organized in one of 3 ways

Single register (Accumulator) organization

o Basic Computer is a good example
o Accumulator is the only general purpose register

General register organization

o Used by most modern computer processors
o Any of the registers can be used as the source or destination for computer operations

Stack organization

o All operations are done using the hardware stack
o 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

The number of address fields in the instruction format depends on the internal organization of CPU.
The three most common CPU organizations:

1**. Single accumulator organization**:
```
ADD    X                                  /* AC ←AC + M[X] */
```
2**. General register organization**:
```
ADD R1, R2, R3        /* R1←R2 + R3 */
ADD R1, R2            /* R1 ←R1 + R2 */
MOVR1, R2             /* R1 ←R2 */
ADD R1, X             /* R1 ←R1 + M[X] */
```
3. Stack organization:
```
PUSHX              /* TOS ←M[X] */
ADD
```

Example:

1. **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*/
```
o Results in short programs

o Instruction becomes long (many bits)

2**. 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 */
```
3. **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 */
```
4. 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
```

Addressing Modes

The addressing mode specifies the rule for translating or modifying the address field of the instruction before the operand is fetched. The way the operands are chosen during program execution is dependent on addressing modes.

computer uses addressing modes :
To give programming versatility by providing a way to implement counters, pointers, indexing of data, and program reallocation.

To reduce number of addressing fields of instruction .

1**. Implied mode:**
The operands are specified implicitly in the definition of the instruction. No need to specify address in the instruction
All register reference instruction in Basic Computer that uses accumulator is from this type. Since registers holding operand(s) are implied in op code of the operation itself.

Zero address instructions in stack-organized computers are implied mode instruction since operands are implied always at top of stack.

Examples from Basic Computer: CLA, CME, INP

2. Immediate mode:

Instead of specifying the address of the operand, operand itself is specified with the instruction.

o No need to specify address in the instruction

o However, operand itself needs to be specified

o Sometimes, require more bits than the address

o Fast to acquire an operand

o Useful mode to initialize registers to constant values (initial).

3. Register mode

Address specified in the instruction is the register address that resides within CPU.

o Designated operand need to be in a register

o Shorter address than the memory address

o Saving address field in the instruction

o Faster to acquire an operand than the memory addressing-

o EA = IR(R) (IR(R): Register field of IR)

4. Register Indirect mode

Instruction specifies a register which contains the memory address of the operand

Saving instruction bits since register address is shorter than the memory address and register is specifying the address here.

Slower to acquire an operand than both the register addressing or memory addressing

User must ensure that address of operand is already sited in mentioned register.

EA = [IR(R)] ([x]: Content of x)

5. Autoincrement/Autodecrement mode

Similar to register indirect mode except that the register is incremented or decremented after or before its value is used to access memory.

Useful to point to next or previous data referenced to current data pointed by register. Therefore used in table access.

Automatically implement Increment/Decrement content of specified register.

6. Direct Address mode

Instruction specifies the memory address which can be used directly to access the memory

o Faster than the other memory addressing modes since operand address comes with op code of the instruction.

o (BAD)Too many bits are needed to specify the address for a large physical memory space.

o In branch type instructions the address field specifies branch address.

o EA = IR(addr) (IR(addr): address field of IR)

7. Indirect Address mode

The address field of an instruction specifies the address of a memory location that contains the address of the operand.

o When the abbreviated address is used large physical memory can be addressed with a relatively small number of bits.

o Slow to acquire an operand because of an additional memory access

o EA = M[IR(address)]

8. Relative Address mode

The Address fields of an instruction specifies the part of the address (abbreviated address) which can be used along with a designated register to calculate the address of the operand

Address field of the instruction is short (few bits)

Large physical memory can be accessed with a small number of address bits

EA = f(IR(address), R), R is sometimes implied .

3 different Relative Addressing Modes depending on R

PC Relative Addressing Mode(R = PC)

EA = PC + IR(address)

Example: if PC=825 and address part in instruction =24. Then address branched to =826 + 24 = 850.

Indexed Addressing Mode(R = IX, where IX: Index Register)

EA = IX + IR(address)

Base Register Addressing Mode(R = BAR, where BAR: Base Address Register) : EA = BAR + IR(address)

9. Indexed Addressing mode

The content of index register is added to address part of instruction to obtain Effective Address (EA).

We can see as address field of instruction specifies the start address of array in memory while index register stores relative position of each entry to start of array.

Some processors have dedicated one CPU register to function as index register. While in other processors (complex ones) have many registers each of them can act as index register.

10. Base Register Addressing mode

The content of base register is added to the address part of the instruction. To obtain EA.

Similar to index addressing except register is called base register instead of index register

The difference can be seen as: base address holds the base address of arrays in memory while address part of instruction holds displacement relative to base register.

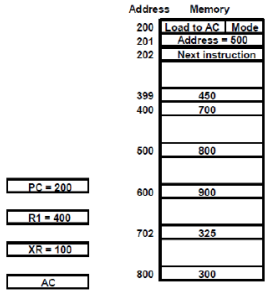
This addressing mode is used facilitate reallocation of programs in memory. When program and data are moved from a segment to another the relative position of data not changed while only its base address will be changed. The change in base register reflects start of new memory segment

EXAMPLE:
Next figure shows a two word instruction at address 200 and 201 that "load to AC". The instruction has an address field occupying second word of value 500.

The first word specifies op code while second word specifies address of operand
o PC= 200 R1 = 400 (register) XR = 100 (index register)

Mode field specifies any mode mentioned earlier

1. In direct mode EA=500 and M[500] = 800. So AC = 800
2. In immediate mode EA=201. Second word of instruction is loaded to AC. So AC = 500
3. In Indirect mode M[500] = 800 which is EA. And M[800] = 300 be loaded into AC. So AC = 300.
4. In relative mode EA=500+202=702. M[702] = 325. So operand=325
5. In index mode EA=XR + 500 = 100 + 500 = 600. M[600] = 900. So AC = 900
6. In register mode operand is in R1. So 400 is loaded into AC. AC = 400
7. In register indirect mode EA=400. M[400] = 700. So AC = 700
8. In auto decrement mode AC= M[R1 – 1] = M[399] = 450



Addressing Mode	Effective Address		Content of AC
Direct address	500	/* AC ← (500) */	800
Immediate operand	-	/* AC ← 500 */	500
Indirect address	800	/* AC ← ((500)) */	300
Relative address	702	/* AC ← (PC+500) */	325
Indexed address	600	/* AC ← (RX+500) */	900
Register	-	/* AC ← R1 */	400
Register indirect	400	/* AC ← (R1) */	700
Autoincrement	400	/* AC ← (R1)+ */	700
Autodecrement	399	/* AC ← -(R) */	450

Data Transfer and Manipulation
Computers provide an extensive set of instructions to give the user the flexibility to execute different tasks.
Instructions sets of different processors differ from ach other in mainly in the way operands are determined from address and mode fields. Even the op-code will be different among them as well.

1. **Data transfer instructions**
Moves data from one place to another. The most common transfer are between memory and processor registers, between processor registers and IO, and between processor registers.

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

Load instruction is used to transfer data from memory to processor register(s) (Accumulator).

Store instruction transfers data from register(s)(Accumulator) to memory.

Move instruction is used to move data from registers and from register to memory and vice versa.

Exchange instruction swaps data between 2 registers or between 2 memory locations.

Input-Output instructions transfer data between processors and IO device

Push-Pop instructions transfer data between stack and registers

Arithmetic Instructions
They will be the 4 basic operations: Add, Subtract, Multiply, and Divide.

Multiplication and Division usually generated using software subroutines.

Next table shows typical arithmetic instructions in general processors.

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, SUB, MUL, DIV instructions may operate with different data types whether available in registers or memory.
o Like in integer type, floating Point type, and BCD type

o Needs special instructions
ADDI add integers

ADDF add floating point

ADDD add in BCD
Since number of bits in registers is finite and hence resulted data are finite precision, some processors support hardware double precision operations arithmetic that occupies 2 words.

3. Logical and Bit manipulation instructions

Logical instructions perform binary operations on bits stored in registers and maybe in memory.

Helpful for manipulating single bits or group of bits

Performs on single bits as separated from each other and treated as Boolean variable.

Clear instruction forces all bits of operand to be 0

o Complement instruction inverts all bits of operand (0 ←1, 1←0).

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

Shift instructions

Shift operands instructions are useful.

Shifts are : logical shifts, arithmetic shifts, and rotate type instructions

Logical shifts insert 0 at the ends .

Arithmetic shifts preserve the sign of operand in most cases but not in all cases (2's complement rule).

Arithmetic shift right preserves sign bit

Arithmetic shift left is the same as logical shift left.

Rotate instructions is a circular shift bit shifted out in one end is inserted in next end.

Rotates instructions may involves carry bit or not.

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 thru carry	RORC
Rotate left thru carry	ROLC

Program Control altered.

The program control instructions may change address in PC and cause the normal sequential execution to be. So this types of instruction causes breaks in execution sequence.

The next table lists some program control instructions.

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RTN
Compare(by –)	CMP
Test(by AND)	TST

Branch and jump instructions are usually the same and they execute immediate change of program sequence to another address.

Branch and Jump instructions may be conditional or unconditional. in conditional case specifies a condition (aero, positive, negative, greater than, and so on) if condition is met the execution transfers to new address. If not met then execution continues sequentially.

Skip instruction skips the instruction immediately after it and executes the next then next one. Conditional skip will do the skip if a condition is met

SKIP ON COND

BRA AD1

BRA AD2

Call and Return instructions are used with subroutines to jump to and come back from subroutines.

Compare instruction subtracts the 2 operand and change some flags in status register. Used to make conditional jumps afterword.

Test instruction performs AND between 2 operands and conditional flags will be changed accordingly.

Status Bit Conditions

ALU of any processor is equipped with condition code bits or flags. Next figure shows 8-bit ALU with 4-bit status flags (C, S, Z, and V). those can be set and cleared.

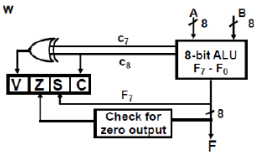
In Basic Computer, the processor had several (status) flags –1 bit value that indicated various information about the processor's state –E, FGI, FGO, I, IEN, R

C (Carry): Set to 1 if the carry out of the ALU is 1

S (Sign): The MSB bit of the ALU"s output

Z (Zero): Set to 1 if the ALU"s output is all 0"s

V (Overflow): Set to 1 if there is an overflow if last 2 carries = 1. If output of ALU > 127 or < -128.



Conditional Branch Instructions

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	Z = 1
BNZ	Branch if not zero	Z = 0
BC	Branch if carry	C = 1
BNC	Branch if no carry	C = 0
BP	Branch if plus	S = 0
BM	Branch if minus	S = 1
BV	Branch if overflow	V = 1
BNV	Branch if no overflow	V = 0
Unsigned compare conditions (A - B)		
BHI	Branch if higher	A > B
BHE	Branch if higher or equal	A ≥ B
BLO	Branch if lower	A < B
BLOE	Branch if lower or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B
Signed compare conditions (A - B)		
BGT	Branch if greater than	A > B
BGE	Branch if greater or equal	A ≥ B
BLT	Branch if less than	A < B
BLE	Branch if less or equal	A ≤ B
BE	Branch if equal	A = B
BNE	Branch if not equal	A ≠ B

Zero bit is used to test if the result of ALU is zero or not

The carry bit is used to test if carry exist in output of ALU or not

The sign bit is used to check if the result is positive or negative. The S bit the most significant bit of result

The overflow bit is used with arithmetic operations done on signed numbers.

For comparison branches a subtract of 2 operands must occur in advance A – B then the comparison branch follows.
CMP A, B CMP A, B CMP A, B CMP A, 0 CMP A, 0

BE BLT BGT BP BN
When S=0 means A >B for signed

When S=1 means A<B for signed

When C=0 means A>B for unsigned

When C=1 means A<B for unsigned

When Z=1 means A=B

Subroutine Call and Return

During execution a subroutine maybe called many times to perform given task at various point of the main program.

A subroutine call transfers control to a subroutine procedure. We can call it
Call subroutine

Jump to subroutine

Branch subroutine

Branch and save return address
When finished subroutine a return instruction returns address back to main program.

in a call subroutine

1. Branch to the beginning of the Subroutine-Same as the Branch or Conditional Branch.

2. Save the Return Address to get the address of the location in the Calling Program upon exit from the Subroutine. Some save return address in: First location, of subroutine. , Fixed location in memory, in fixed register. In stack.

Micro operations implementing calling/returning from subroutines

```
CALL
  SP ← SP - 1
  M[SP] ← PC
  PC ← EA

RTN
  PC ← M[SP]
  SP ← SP + 1
```

Reference

1.‘Computer System Architecture’, Morris M. Mano, 3rd edition,Prentice Hall India.

2. Computer Organization and Achitecture, William Stallings ,8th edition,PHI

3. Computer Organization, Carl Hamachar, Vranesic, McGraw Hill.