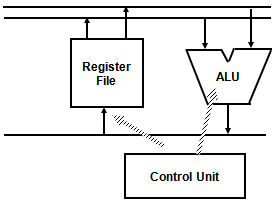
**MAJOR COMPONENTS OF CPU**

Storage Components:

Registers

Flip-flops

Execution (Processing) Components:

Arithmetic Logic Unit (ALU):

Arithmetic calculations, Logical computations, Shifts/Rotates

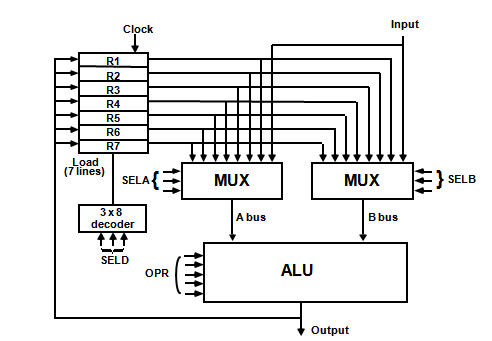
Transfer Components:

Bus

Control Components:

Control Unit

**GENERAL REGISTER ORGANIZATION**

****

The control unit directs the information flow through ALU by:

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

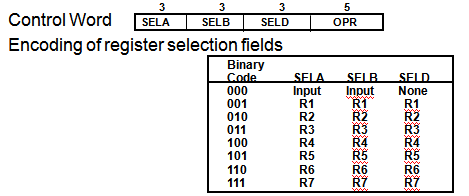
Example: R1 <- R2 + R3

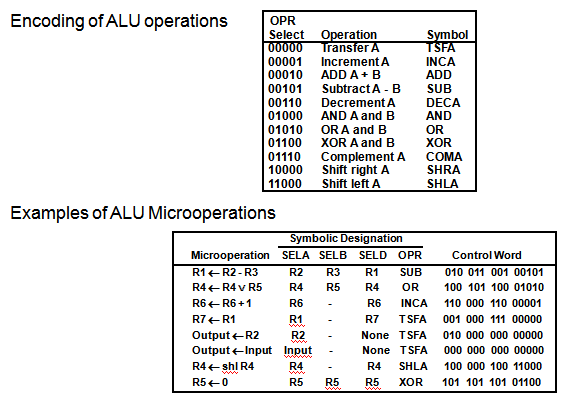
[1] MUX A selector (SELA): BUS A ← R2

[2] MUX B selector (SELB): BUS B ← R3

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

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

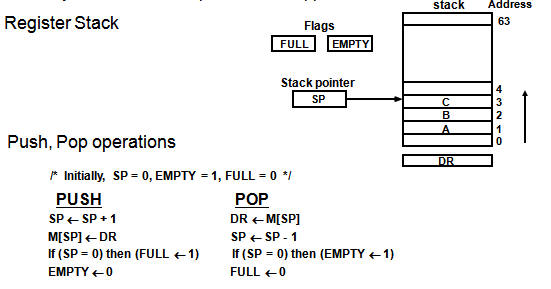




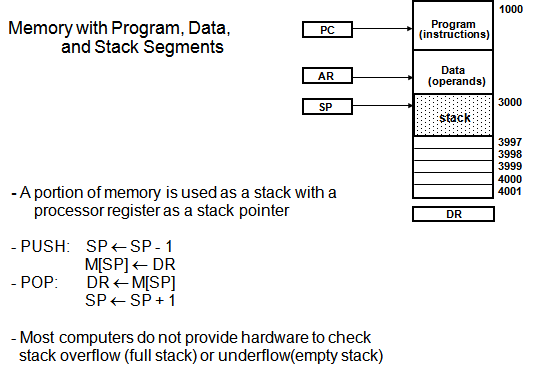
**REGISTER STACK ORGANIZATION**

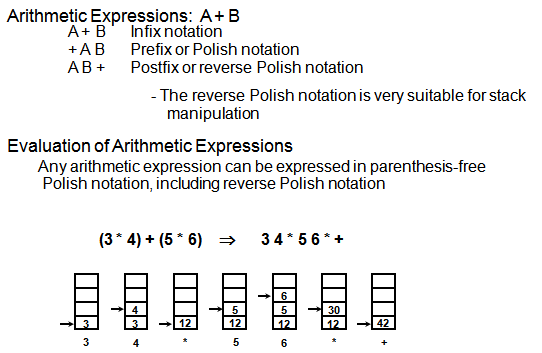
Stack

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



MEMORY STACK ORGANIZATION





**INSTRUCTION FORMAT**

Instruction Fields

OP-code field - specifies the operation to be performed

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

Mode field - specifies the way the operand or the effective address is determined

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

**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 \*/

MOV R1, R2 /\* R1 ← R2 \*/

ADD R1, X /\* R1 ← R1 + M[X] \*/

**Stack organization:**

PUSH X /\* TOS ← M[X] \*/

ADD

**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[B] \*/

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 \*/

**ADDRESSING MODES**

Addressing Modes:

\* Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced)

\* Variety of addressing modes

- to give programming flexibility to the user

- to use the bits in the address field of the instruction efficiently

**Implied Mode**

Address of the operands are specified implicitly in the definition of the instruction

- No need to specify address in the instruction

- EA = AC, or EA = Stack[SP**], EA: Effective Address.**

**Immediate Mode**

Instead of specifying the address of the operand, operand itself is specified

- No need to specify address in the instruction

- However, operand itself needs to be specified

- Sometimes, require more bits than the address

- Fast to acquire an operand

**Register Mode**

Address specified in the instruction is the register address

- Designated operand need to be in a register

- Shorter address than the memory address

- Saving address field in the instruction

- Faster to acquire an operand than the memory addressing

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

**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

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

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

**Auto-increment or Auto-decrement features:**

Same as the Register Indirect, but:

- When the address in the register is used to access memory, the value in the register is incremented or decremented by 1 (after or before the execution of the instruction)

**Direct Address Mode**

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

- Faster than the other memory addressing modes

- Too many bits are needed to specify the address for a large physical memory space

- EA = IR(address), (IR(address): address field of IR)

**Indirect Addressing Mode**

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

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

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

- EA = M[IR(address)]

**Relative Addressing Modes**

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

PC Relative Addressing Mode(R = PC)

- EA = PC + IR(address)

- Address field of the instruction is short

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

**Indexed Addressing Mode**

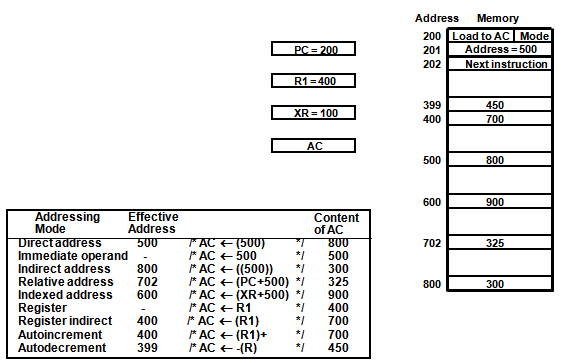
XR: Index Register:

- EA = XR + IR(address)

**Base Register Addressing Mode**

BAR: Base Address Register:

- EA = BAR + IR(address)



**DATA TRANSFER INSTRUCTIONS**

**DATA MANIPULATION INSTRUCTIONS**

**PROGRAM CONTROL INSTRUCTIONS**

**CONDITIONAL BRANCH INSTRUCTIONS**

**SUBROUTINE CALL AND RETURN**

**PROGRAM INTERRUPT**

External interrupts

External Interrupts initiated from the outside of CPU and Memory

- I/O Device -> Data transfer request or Data transfer complete

- Timing Device -> Timeout

- Power Failure

Internal interrupts (traps)

Internal Interrupts are caused by the currently running program

- Register, Stack Overflow

- Divide by zero

- OP-code Violation

- Protection Violation

Software Interrupts

Both External and Internal Interrupts are initiated by the computer Hardware.

Software Interrupts are initiated by executing an instruction.

- Supervisor Call -> Switching from a user mode to the supervisor mode

-> Allows to execute a certain class of operations which are not allowed in the user mode

INTERRUPT PROCEDURE

- The interrupt is usually initiated by an internal or an external signal rather than from the execution of an instruction (except for the software interrupt)

- The address of the interrupt service program is determined by the hardware rather than from the address field of an instruction

- An interrupt procedure usually stores all the information necessary to define the state of CPU

rather than storing only the PC. The state of the CPU is determined from; Content of the PC

Content of all processor registers, Content of status bits. many ways of saving the CPU state depending on the CPU architectures.

**RISC: REDUCED INSTRUCTION SET COMPUTERS**

**Characteristics of CISC:**

* A large number of instructions (from 100-250 usually)
* Some instructions that performs a certain tasks are not used frequently.
* Many addressing modes are used (5 to 20)
* Variable length instruction format.
* Instructions that manipulate operands in memory.

**PHYLOSOPHY OF RISC**

**Reduce the semantic gap between**

**machine instruction and microinstruction**

**1-Cycle instruction**

Most of the instructions complete their execution in 1 CPU clock cycle - like a microoperation

\* Functions of the instruction (contrast to CISC)

- Very simple functions

- Very simple instruction format

- Similar to microinstructions

=> No need for microprogrammed control

\* Register-Register Instructions

- Avoid memory reference instructions except Load and Store instructions

- Most of the operands can be found in the registers instead of main memory

=> Shorter instructions

=> Uniform instruction cycle

=> Requirement of large number of registers

\* Employ instruction pipeline

Operations are register-to-register, with only LOAD and STORE accessing memory The operations and addressing modes are reduced Instruction formats are simple.