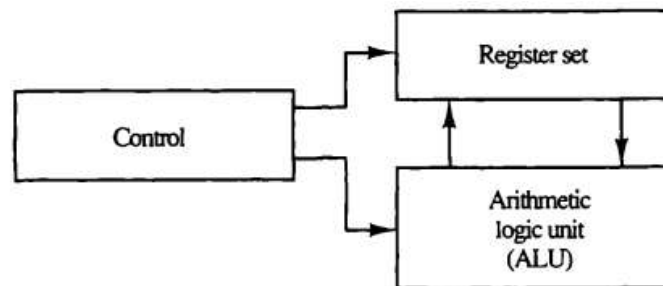


Central Processing Unit



- The part of the computer that performs the bulk of data-processing operations is called the central processing unit and is referred to as the CPU. The CPU is made up of three major parts: Register set, ALU and Control unit. The register set stores intermediate data used during the execution of the instructions. The arithmetic logic unit (ALU) performs the required microoperations 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.
- The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer. Computer architecture is sometimes defined as the computer structure and behavior as seen by the programmer that uses machine language instructions. This includes the instruction formats, addressing modes, the instruction set, and the general organization of the CPU registers.



• General Register Organization

Memory locations are needed for storing pointers, counters, return addresses, temporary results, and partial products during multiplication. Having to refer to memory locations for such applications is time consuming because memory access is the most time-consuming operation.

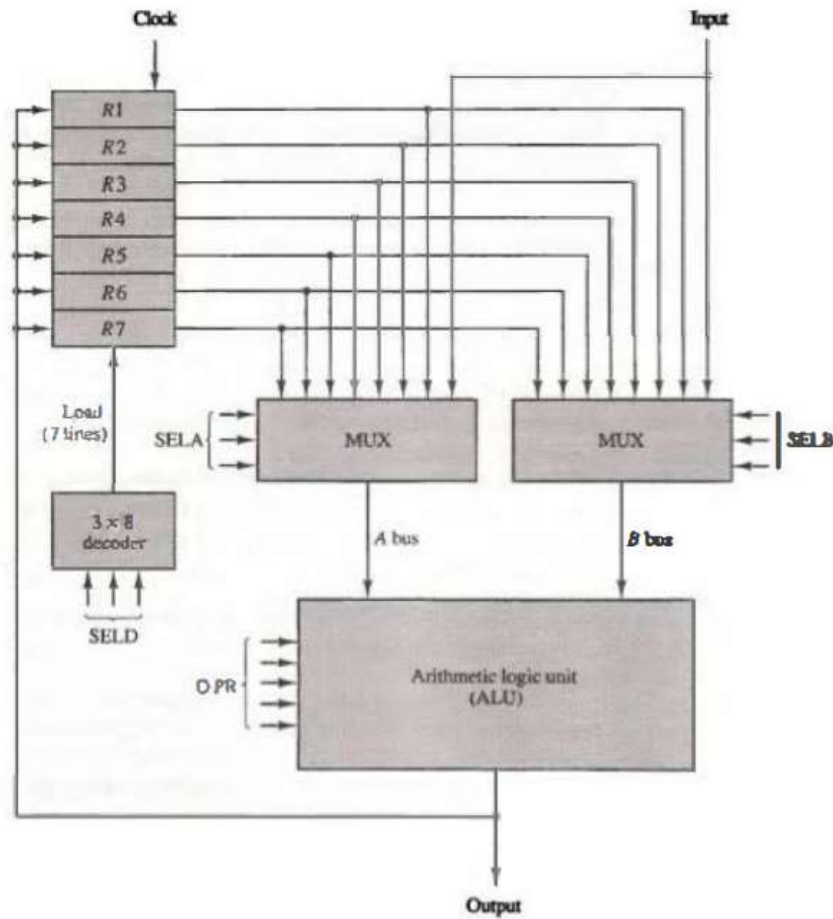
It is more convenient and more efficient to store these intermediate values in processor registers. When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system.

The registers communicate with each other not only for direct data transfers, but also while performing various microoperations. Hence it is necessary to provide a common unit that can perform all the arithmetic, logic, and shift microoperations in the processor

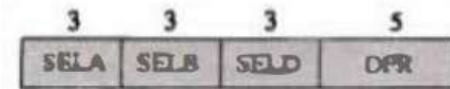
- **General Register Organization**

- A bus organization for seven CPU registers is shown in Fig on the next slide. 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 microoperation that is to be performed. The result of the microoperation 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.

CPU Block diagram



(a) Block diagram



(b) Control word

- 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 $R1 \leftarrow R2 + R3$

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 of R3 into bus B.
- 3. ALU operation selector (OPR): to provide the arithmetic addition $A + B$.
- 4. Decoder destination selector (SELD): to transfer the content of the output bus into R1.
- The four control selection variables are generated in the control unit and must be available at the beginning of a clock cycle. 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. Then, when the next clock transition occurs, the binary information from the output bus is transferred into R1

- A useful feature that is included in the CPU of most computers is a stack or last-in, first-out (LIFO) list. A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The operation of a stack can be compared to a stack of trays. The last tray placed on top of the stack is the first to be taken off.
- The stack in digital computers is essentially 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. Contrary to a stack of trays where the tray itself may be taken out or inserted, 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.

- **Two operations of Stack are:**

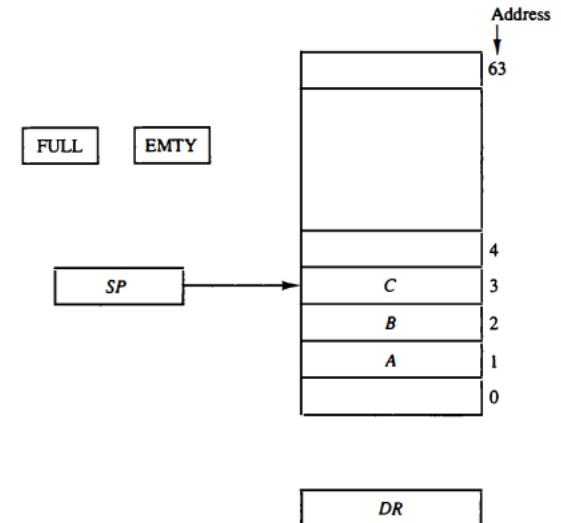
PUSH: The operation of insertion is called push (or push-down) because it can be thought of as the result of pushing a new item on top.

POP: 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.

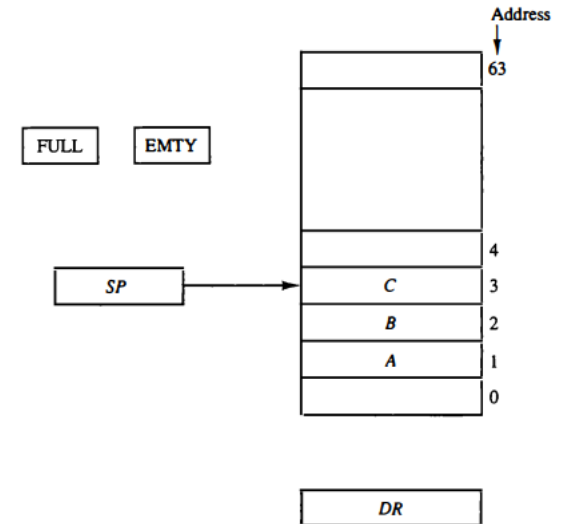
However, nothing is pushed or popped in a computer stack. These operations are simulated by incrementing or decrementing the stack pointer register.

Register Stack

- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure shows the organization of a 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.
- Item B is now on top of the stack since SP holds address 2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack. Note that item C has been read out but not physically removed. This does not matter because when the stack is pushed, a new item is written in its place



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). When 63 is incremented by 1, the result is 0 since $111111 + 1 = 1000000$ 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, so that SP points to the word at address 0 and the stack is marked empty and not full. 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 microoperations;

- **SP <- SP + 1 Increment stack pointer**
- **M[SP] <- DR Write item on top of the stack pop**
- **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. Note that 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. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.

- POP Operation: A new item is deleted from the stack if the stack is not empty (if $EMPTY = 0$). The pop operation consists of the following sequence of microoperations:
 - $DR \leftarrow M[SP]$ Read item from the top of stack
 - $SP \leftarrow SP - 1$ Decrement stack pointer.
 - If $(SP = 0)$ then $(EMPTY \leftarrow 1)$ Check if stack is empty.
 - $FULL \leftarrow 0$ 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 EMPTY 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. Note that 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 $EMPTY = 1$.