

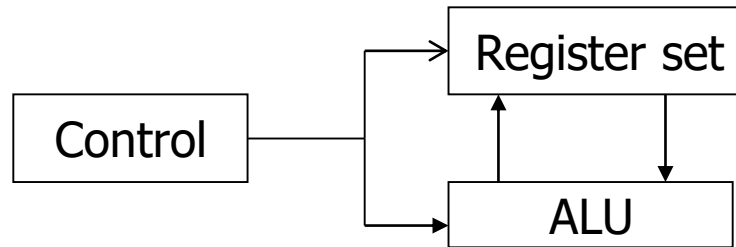
Unit 2

CPU Architecture types

M. Morris Mano (Ch. 8)

Central Processing Unit

Introduction



- The CPU is made up of 3 major Components.
- The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer
- In programming, memory locations are needed for storing pointers, counters, return addresses, temporary result , etc. Memory access is most time consuming operation in a computer.
- It is then more convenient and more efficient to store these intermediate values in processor registers, which are connected through common bus system.

CPU Organization

Accumulator based CPU (Govind Rajlu 3.3.1)



Characteristics :

Initially, computers had accumulator based CPUs.

It is a simple CPU in which the accumulator contains an operand for the Instruction.

The instruction leaves the result in the accumulator.

These CPUs have zero address & single address instruction.

The Advantages :

Short Instruction & less memory space.

Instruction cycle takes less time because it saves time in Instruction fetching due to the absence of operand fetch.

The Disadvantages :

Program Size increases, memory size increases.

Program execution time increases due to increase in program size.



CPU Organization

Register based CPU (Govind Rajlu 3.3.2)

Characteristics:

Multiple registers are used.

The use of registers result in short programs with limited instructions.

The Advantages :

Shorter Program size

Increase in the number of registers, increases CPU efficiency.

The Disadvantages :

Additional memory accesses are needed.

Compilers need to be more efficient in this aspect



Central Processing Unit

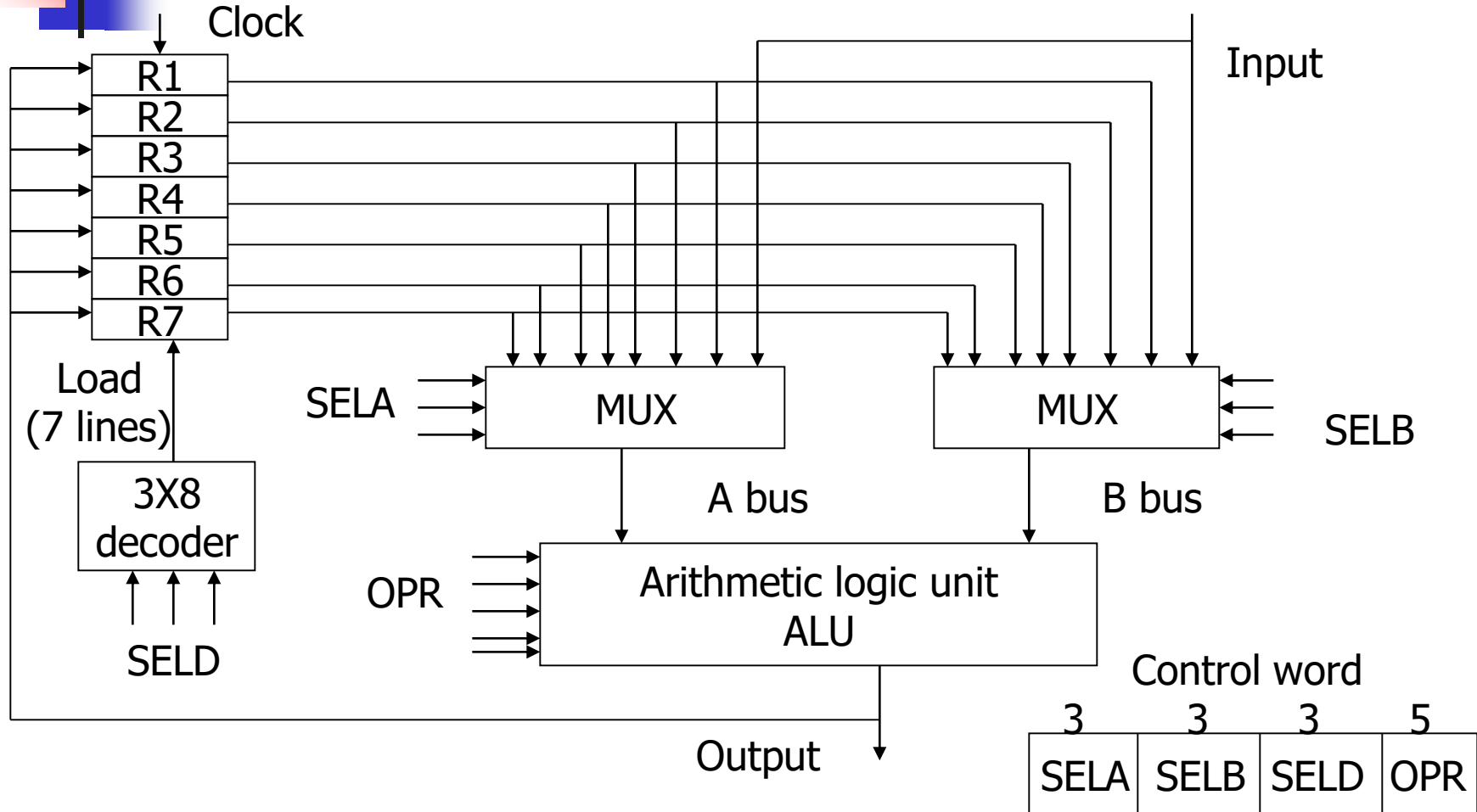
General Register Organization (Morris Mano Ch 8)

The design of a CPU is a task that involves choosing the hardware for implementing the machine instructions. ■

Lets describe how the registers communicate with the ALU through buses and explain the operation of the memory stack. ■

Central Processing Unit

Detailed data path of a register based CPU



Example of Microoperations:


$$R1 \leftarrow R2 + R3$$

- To Perform this operation, 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**.
- Decoder destination selector (**SELD**): to transfer the content of the output bus into **R1**.
- There are therefore **14** binary selection inputs , and their combination value specifies a **control word** (**See Tables 8-1, 8-2 & 8-3 (Morris Mano)**)
- A control Word (**CW**) is a word whose individual bits represent a various control signals.



Central Processing Unit

General Register Organization

- A complete CPU :
- A powerful CPU can be designed using a structure with an **instruction unit** that fetch instructions from an instruction cache or from the main memory if the desired instructions are not in the cache.
- It has also separate processing units to deal with integer data and floating point data.
- A data cache is inserted between these units and the main memory.
- Other processor use a single cache (data and instruction) that store both instructions and data.



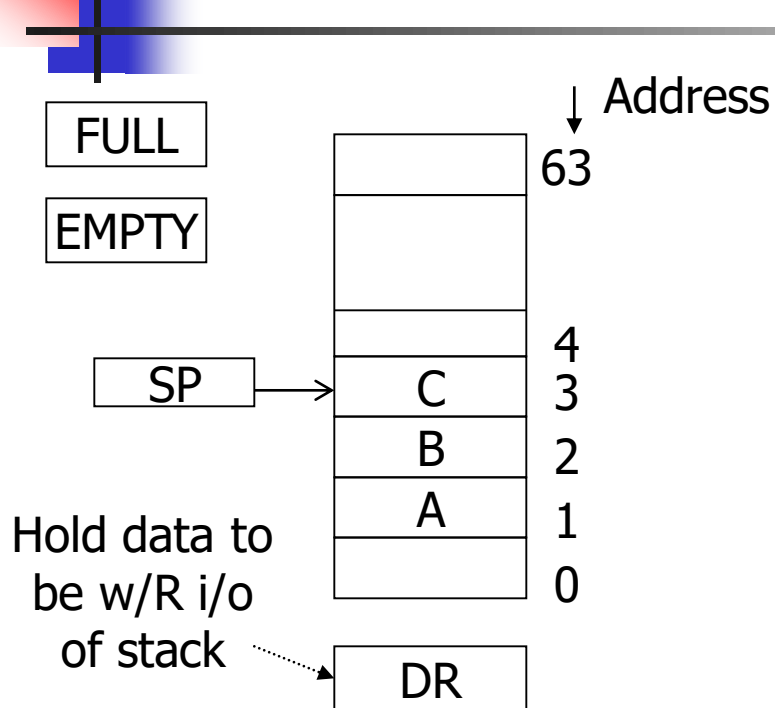
Central Processing Unit

Stack Organization

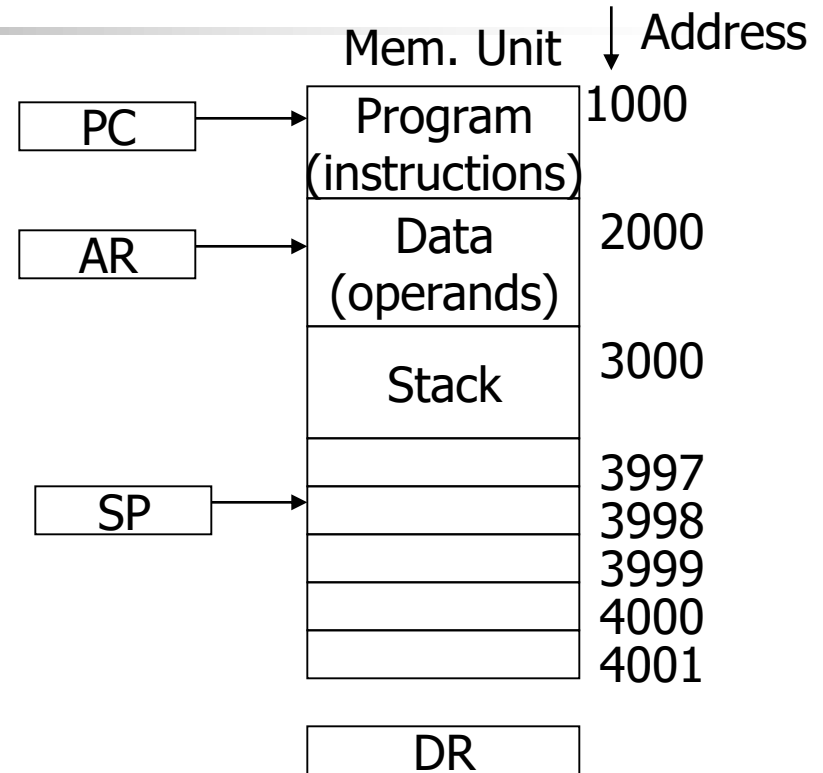
- The stack in digital computers is essentially a memory unit with an address register (Stack Pointer **SP**) that count only after an initial value is loaded into the stack.
- **SP** value always point at the top item in the stack.
- The **2** operations of stacks are the insertion (**push**), and deletion (**pop**) of items.
- A stack can be organized as a collection of a finite number of memory words or registers.
- In a 64-word stack, SP contains 6 bits because $2^6 = 64$.
- The 1-bit register FULL is set to 1 when stack is full.
- The 1-bit register EMTY is set to 1 when stack is empty.
- DR is data register that holds the binary data to be written into or read out of the stack.

Central Processing Unit

Stack Organization



Block diagram of
a 64 word-register stack
6 bit SP



Computer memory with
program, data & stack segments

Central Processing Unit

Stack Organization

In register stack

- The **push** operation is implemented with the following sequence of microoperations:

$SP \leftarrow SP + 1$ Increment stack pointer

$M[SP] \leftarrow DR$ Write item on top of the stack

If $(SP = 0)$ then $(FULL \leftarrow 1)$ Check if stack is full

(when 63 is incremented by 1, the result is 0.)

$EMPTY \leftarrow 0$ Mark the stack not empty

Central Processing Unit

Stack Organization

In register stack

- The **pop** operation consists the following sequence of microoperations:

$DR \leftarrow M[SP]$ Read item from top of the 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

A stack can exist as a stand-alone unit or can be implemented in a RAM attached to a CPU.



Central Processing Unit

Stack Organization

In Memory stack

- A stack can be implemented in a portion of a random –access memory attached to a CPU, and using a processor register as **SP** as shown the diagram above.
- In the diagram shown Computer memory with program, data & stack segments:
 - PC (points at the address of the next instruction) is used during the **fetch** phase to read an instruction.
 - AR (points at an array of data) is used during the **execution** phase to read an operand.
 - SP is used **push** or **pop** items into or from the stack
- The three registers are connected to a common address bus, & either one can provide an address for memory.
- **Push**: insert (write) new item at the top of the stack
$$SP \leftarrow SP - 1, \quad M[SP] \leftarrow DR$$
- **Pop**: delete (read) the top item from the stack
$$DR \leftarrow M[SP], \quad SP \leftarrow SP + 1$$

Central Processing Unit

Stack Organization

In Memory stack

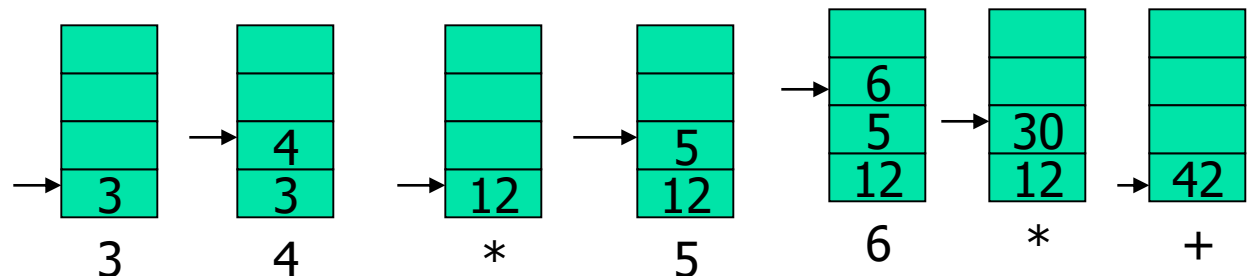
- The stack limits (overflow) can be checked by using **2** processor registers:
 - One to hold the upper limit (3000 in this example)
 - Another to hold the lower limit (4001 in this example)
- **SP** is compared with the **upper limit** register after each **push** operation,
- **SP** is compared with the **lower limit** register after each **pop** operation.
- *How many **microoperations** are needed then for the **pop** or **push** operations?*
- **2 microoperations** are needed for push or pop :
 1. An access to memory through **SP**
 2. Updating **SP**

Central Processing Unit

Stack Organization

Reverse Polish Notation

- A stack organization is very effective for evaluating arithmetic expressions.
- The reverse Polish notation is in a form suitable for stack manipulation, where the infix expression $A*B+C*D$ can be written in Reverse Polish Notation (RPN or postfix notation) as: $AB*CD*+$
- Example: Stack operation to evaluate $3*4+5*6$
 $\Rightarrow 34*56*+$ (in reverse Polish notation)





The advantages of stack based CPU:

Easy Programming /high compiler Efficiency.
Instructions don't have address field, short instructions.

The disadvantages :

Additional hardware circuitry needed for stack implementation.
Increased program size.



Central Processing Unit

Instruction Formats

- The bits of the instruction are divided into groups called fields.
- The most common fields founded in the instruction formats are:
 - 1. An operation code field
 - 2. An address field.
 - 3. A mode field (Addressing modes, Sec. 8.5)
- Data executed by instructions (operands) are stored either in memory or in processor registers.
- Operands residing in memory are specified by their memory address.
- Operands residing in registers are specified with a register address.



Central Processing Unit

Instruction Formats

- A register address is binary number of k bits that defines one of 2^k registers in the CPU.
- Most computers fall into one of the 3 types of CPU organizations:
 1. Single Accumulator (AC) Organization, i.e. `ADD X`
 2. General register (Rs) Organization, `ADD R1,R2,R3`
 3. Stack Organization, i.e. `ADD` (pop and add 2 operand then push the result into the stack)
- Some computers combine features from more than one organization structure, Ex. Intel 8080 (GRs for register transfer, AC used in arithmetic operations)



Central Processing Unit

Instruction Formats

- Example: the influence of the number of addresses on the way of evaluating this arithmetic statement using different:

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

- 3-address instructions: [see Page 260](#)
- 2-address instructions: [see Page 260](#)
- 1-address instructions: [see Page 261](#)
- Zero-address instructions: [see Page 261](#)



Central Processing Unit

Addressing modes

- The addressing mode specifies a rule for interpreting or modifying the **address field** of the instruction before the operand is actually executed.
- Computers use addressing mode techniques for the purpose of accommodating one of the following provisions:
 1. To give programming versatilities to the user to be more flexible.
 2. To reduce the number of bits in the addressing field of the instruction.
- In some some computers, the addressing mode of the instruction is specified with **distinct binary code**.

Instruction format with mode field

Opcode	Mode	Address
--------	------	---------



Central Processing Unit

Addressing modes

- Other computers use a **single binary** for operation & Address mode.
- The mode field is used to **locate the operand**.
- Address field may designate a memory address or a processor register.
- There are 2 modes that need no address field at all (**Implied & immediate modes**).



Central Processing Unit

Addressing modes

- The most well known addressing mode then are:
 - Implied mode.
 - Immediate mode
 - Register mode
 - Register Indirect mode
 - Auto-increment or Auto-decrement mode
 - Direct Mode
 - Indirect Mode
 - Relative Address Mode
 - Index Addressing Mode

Central Processing Unit

Addressing modes

Numerical Example

PC=200		R1=400		Address	Memory		
XR=100		AC			200	Load to AC	Mode
					201	Address=500	
				202	Next Instruction		
Addressing mode		eff. Add	Content of AC				

Direct Address	500	800	399	450			
Immediate operand	201	500	400	700			
Indirect Address	800	300	500	800			
Relative Address	702 (PC=PC+2)	325					
Indexes Address	600 (XR+500)	900	600	900			
Register	---	400	702	325			
Register Indirect	400	700					
Auto-increment	400	700	800	300			
Auto-decrement	399	450					

Tabular list



Central Processing Unit

Addressing modes

- Different ways in which the address of an operand is specified in an instruction is referred to as addressing modes.
- **Register mode**
 - Operand is the contents of a processor register.
 - Address of the register is given in the instruction.
 - E.g. *Clear R1*
- **Absolute mode**
 - Operand is in a memory location.
 - Address of the memory location is given explicitly in the instruction.
 - E.g. *Clear A*
 - Also called as “Direct mode” in some assembly languages
- **Register and absolute modes can be used to represent variables**

Central Processing Unit

Addressing modes

Immediate mode

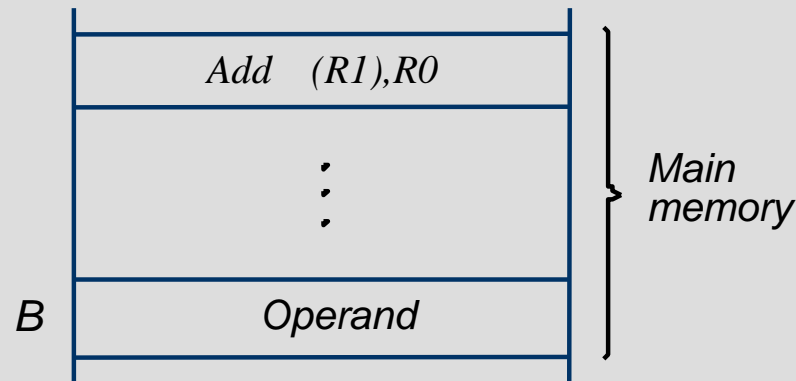
- Operand is given explicitly in the instruction.
 - E.g. *Move #200, R0*
 - Can be used to represent constants.
- Register, Absolute and Immediate modes contained either the address of the operand or the operand itself.
- Some instructions provide information from which the memory address of the operand can be determined
 - That is, they provide the “Effective Address” of the operand.
 - They do not provide the operand or the address of the operand explicitly.
- Different ways in which “Effective Address” of the operand can be generated.

Central Processing Unit

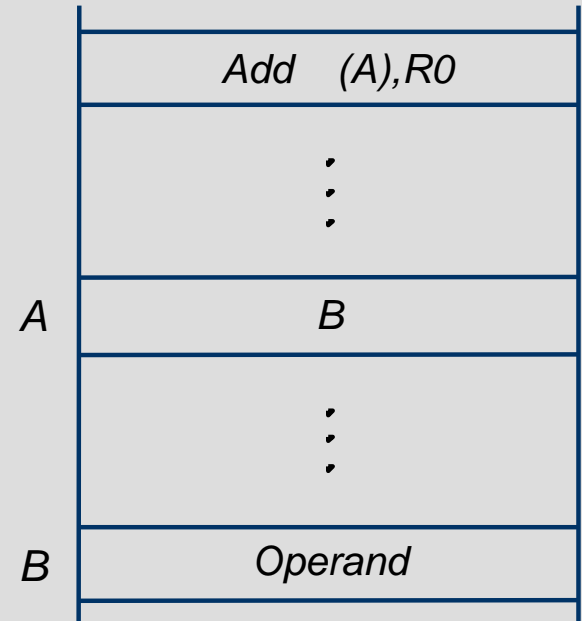
Addressing modes

Indirect Mode

R1 and A are called “pointers”



- Register R1 contains Address B
- Address B has the operand



- Address A contains Address B
- Address B has the operand

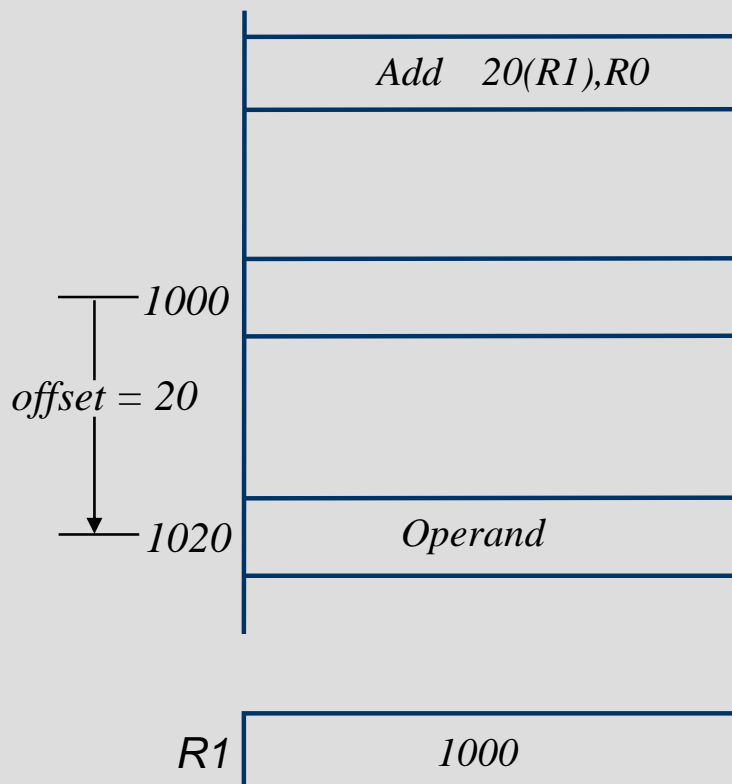
Effective Address of the operand is the contents of a register or a memory location whose address appears in the instruction.

Central Processing Unit

Addressing modes

Indexing Mode

Effective Address of the operand is generated by adding a constant value to the contents of the register



- *Operand is at address 1020*
- *Register R1 contains 1000*
- *Offset 20 is added to the contents of R1 to generate the address 20*
- *Contents of R1 do not change in the process of generating the address*
- *R1 is called as an “index register”*

What address would be generated by `Add 1000(R1), R0` if R1 had 20?

Central Processing Unit

Addressing modes

Relative mode

- Effective Address of the operand is generated by adding a constant value to the contents of the Program Counter (PC).
- Variation of the Indexing Mode, where the index register is the PC instead of a general purpose register.
- When the instruction is being executed, the PC holds the address of the next instruction in the program.
- Useful for specifying target addresses in branch instructions.

Addressed location is “relative” to the PC, this is called “Relative Mode”

Central Processing Unit

Addressing modes

■ Autoincrement mode:

- Effective address of the operand is the contents of a register specified in the instruction.
- After accessing the operand, the contents of this register are automatically incremented to point to the next consecutive memory location.
- $(R1)+$

■ Autodecrement mode

- Effective address of the operand is the contents of a register specified in the instruction.
- Before accessing the operand, the contents of this register are automatically decremented to point to the previous consecutive memory location.
- $-(R1)$

- Autoincrement and Autodecrement modes are useful for implementing “Last-In-First-Out” data structures