Course:
Computer Architecture

Paper:
MCA – 202

Module:
General Organization
Lecture - 1

Outline

- Introduction
- General Register Organization
  - Control Word
  - Microoperations
- Stack Organization
- CPU Organization
- Addressing Modes
- Interrupt

Introduction:

- Part of computer that performs bulk of data processing operations

- CPU made of three major parts:
  - Register set
  - ALU
  - Control Unit

Inside the CPU

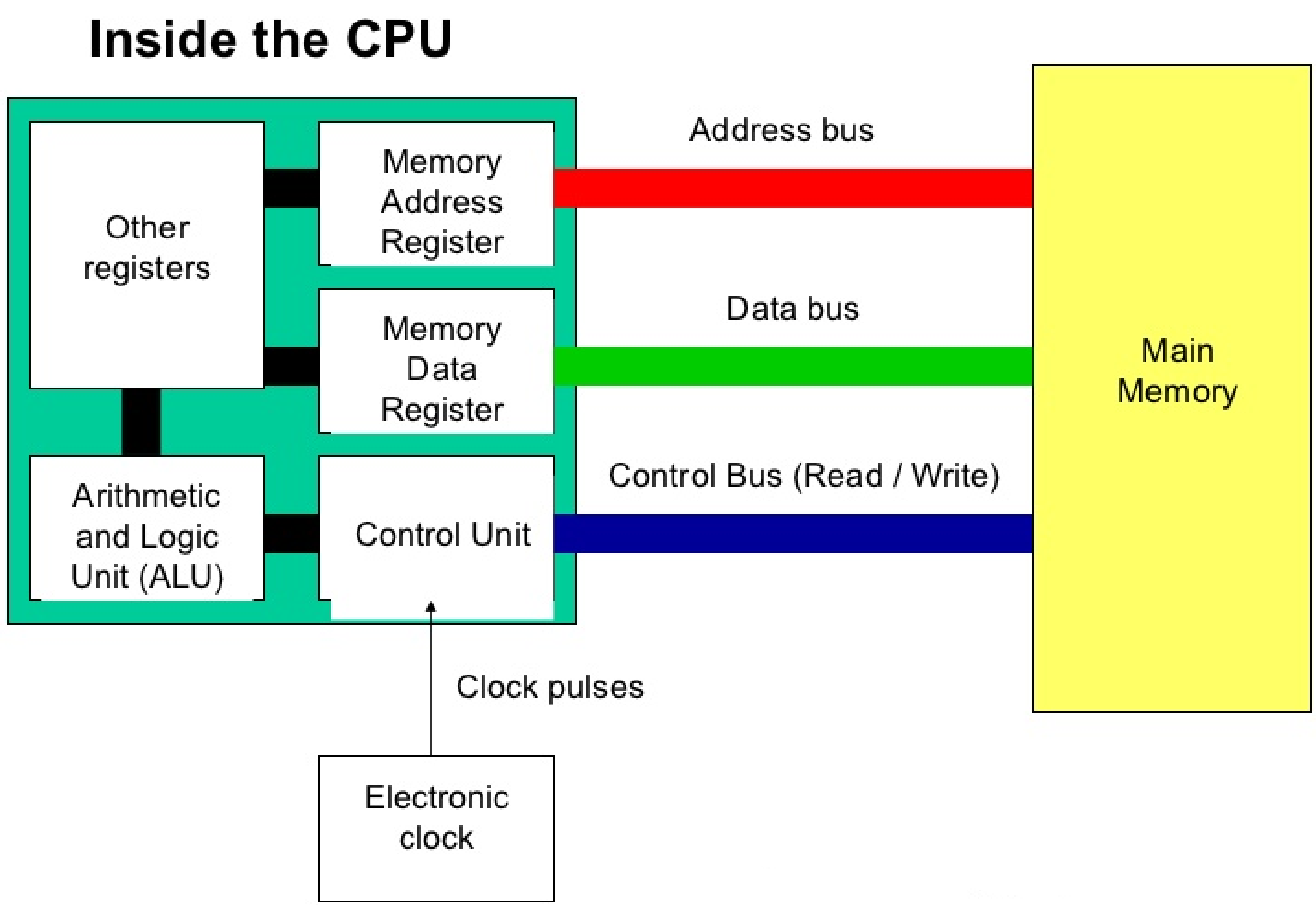
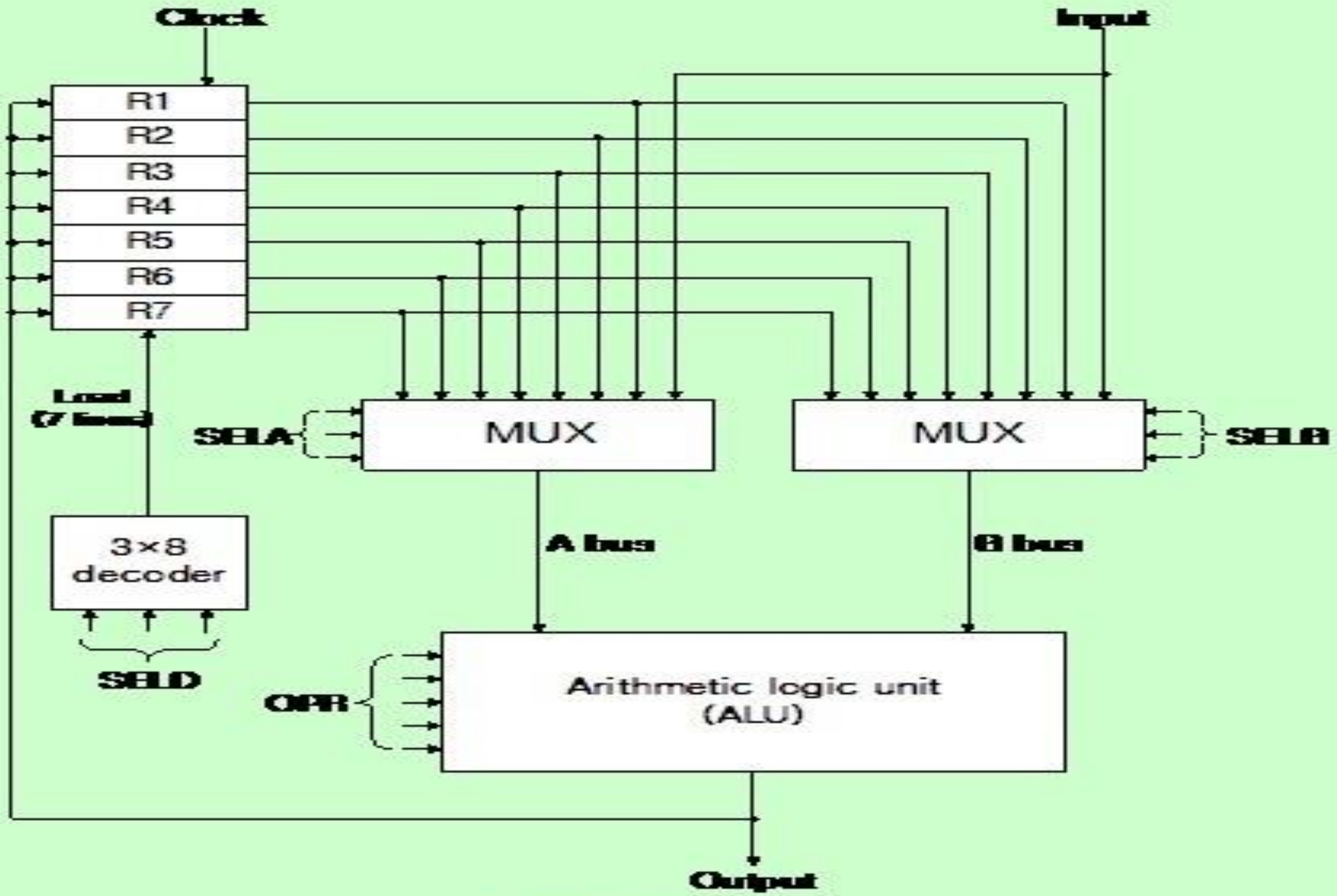


Image Courtesy: Google

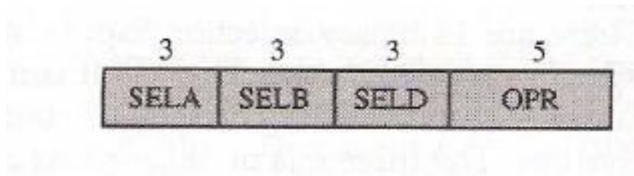


(a) Block diagram

3	3	3	5
SELA	SELB	SELD	OPR

(b) Control word

Control Word:



Control Word

- For ex:

$$R1 \leftarrow R2 + R3$$

  - MUX A selector (SELA): to place the content of R2 into bus A
  - MUX B selector (SELB): to place the content of R3 into bus B
  - ALU operation selector (OPR): to provide arithmetic addition A+B
  - Decoder destination selector (SELD): to transfer the content of the output bus into R1

Operations:

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*

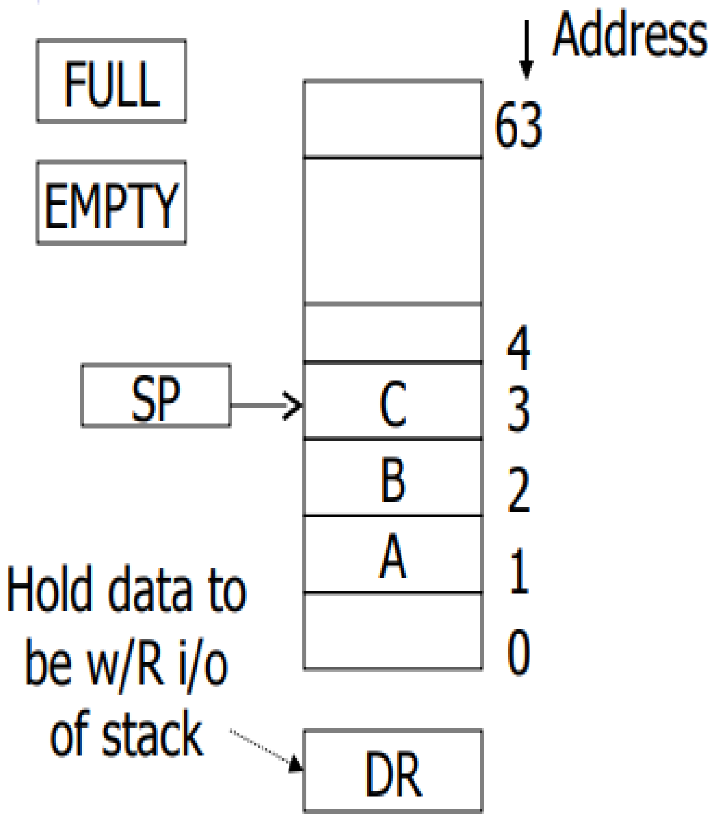
Microoperations:

Micro operation	SELA	SELB	SELD	OPR	Control Word
R1 ⟵ R2 – R3	R2	R3	R1	SUB	010 011 001 00101
R4 ⟵ R4 v R5	R4	R5	R4	OR	100 101 100 01010
R6 ⟵ R6 + 1	R6	-	R6	INCA	110 000 110 00001
R7 ⟵ R1	R1	-	R7	TSFA	001 000 111 00000
Output ⟵ R2	R2	-	None	TSFA	010 000 000 00000
Output ⟵ Input	Input	-	None	TSFA	000 000 000 00000
R4 ⟵ sh1 R4	R4	-	R4	SHLA	100 000 100 11000
R5 ⟵ 0	R5	R5	R5	XOR	101 101 101 01100

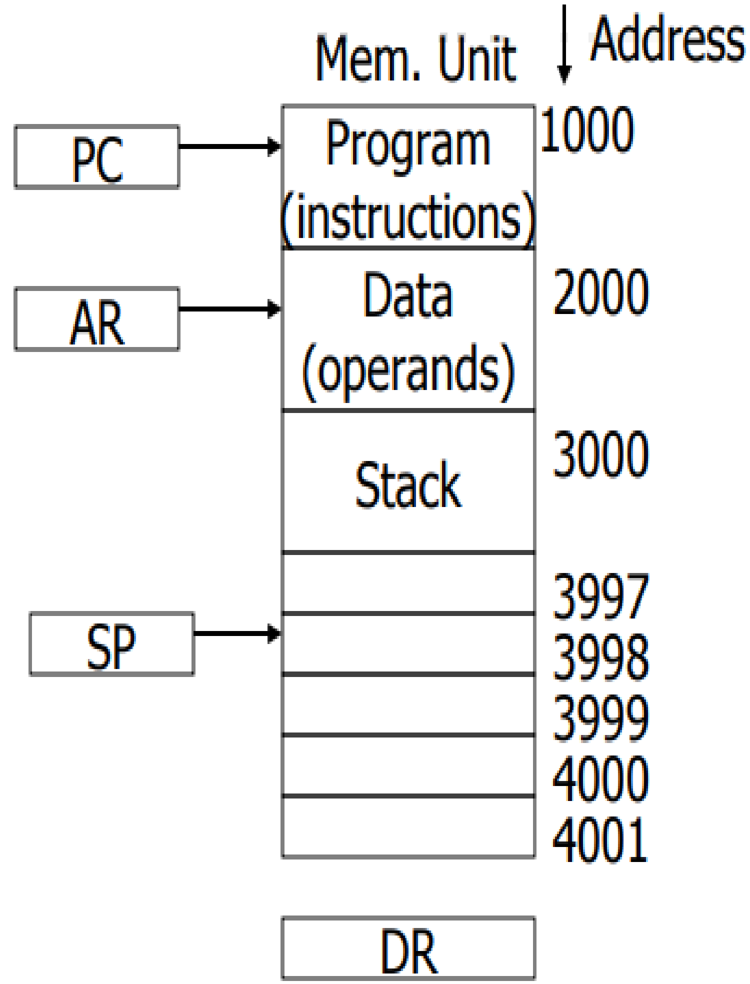
Continued…

- Most efficient way to generate control words with large number of bits is to store them in memory unit
- A memory unit that stores control words is referred control memory
- By reading consecutive control words from memory, it is possible to initiate desired sequence of micro-operations
- This is referred as microprogrammed control

Stack Organization



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



Computer memory with
program, data & stack segments

PUSH

- The PUSH operation is implement 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

POP

- 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.

CPU Organization

- 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)

Address Instruction

- Three-Address Instruction
  - ADD R1, A, B R1 ← M[A] + M[B]
  - ADD R2, C, D R2 ← M[C] + M[D]
  - ADD X, R1, R2 M[X] ← R1 * R2
- Two-Address Instruction
  - 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

Continued…

- One-Address Instruction
  - LOAD A   AC ← M[A]
  - ADD B   AC ← AC + M[B]
  - STORE   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

- Zero-Address Instruction
  - PUSH A
  - PUSH B
  - ADD
  - PUSH C
  - PUSH D
  - ADD
  - MUL
  - POP X

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 computers, the addressing mode of the instruction is specified with distinct binary code.

Instruction format with mode field

Opcode	Mode	Address

Continued…

- 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).

Different addressing mode

- Implied mode
  - Operands are specified implicitly in the definition of the instruction
    - All register reference that use an accumulator, Stack instruction
- Immediate mode
  - Operands specified in the instruction itself
- Register mode
  - Operands are in register
- Register Indirect mode
  - Specifies a register in CPU whose content gives the address of the operand in the memory

Different addressing mode

- Autoincrement or Autodecrement mode
  - Register indirect mode whose value increases or decreases after its value is used to access memory
- Direct Address mode
  - Effective address is equal to the address part of the instruction
- Indirect Address mode
  - Address field of the instruction gives the address where effective address is stored in memory
- Relative Address mode
  - Content of the program counter is added to the address part of the instruction to obtain effective address

Different addressing mode

- Indexed Addressing mode
  - Content of an index register is added to the address part of the instruction to obtain effective address
- Base Addressing mode
  - Content of an base register is added to the address part of the instruction to obtain effective address

Tabular List:

PC=200	R1=400
XR=100	AC

Addressing mode	eff. Add	Content of AC
Direct Address	500	800
Immediate operand	201	500
Indirect Address	800	300
Relative Address	702(PC=PC+2)	325
Indexes Address	600(XR+500)	900
Register	---	400
Register Indirect	400	700
Auto-increment	400	700
Auto-decrement	399	450

Address	Memory	
200	Load to AC	Mode
201	Address=500	
202	Next Instruction	
399	450	
400	700	
500	800	
600	900	
702	325	
800	300	

Addressing Modes:

- Different ways in which the address of an operand in 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

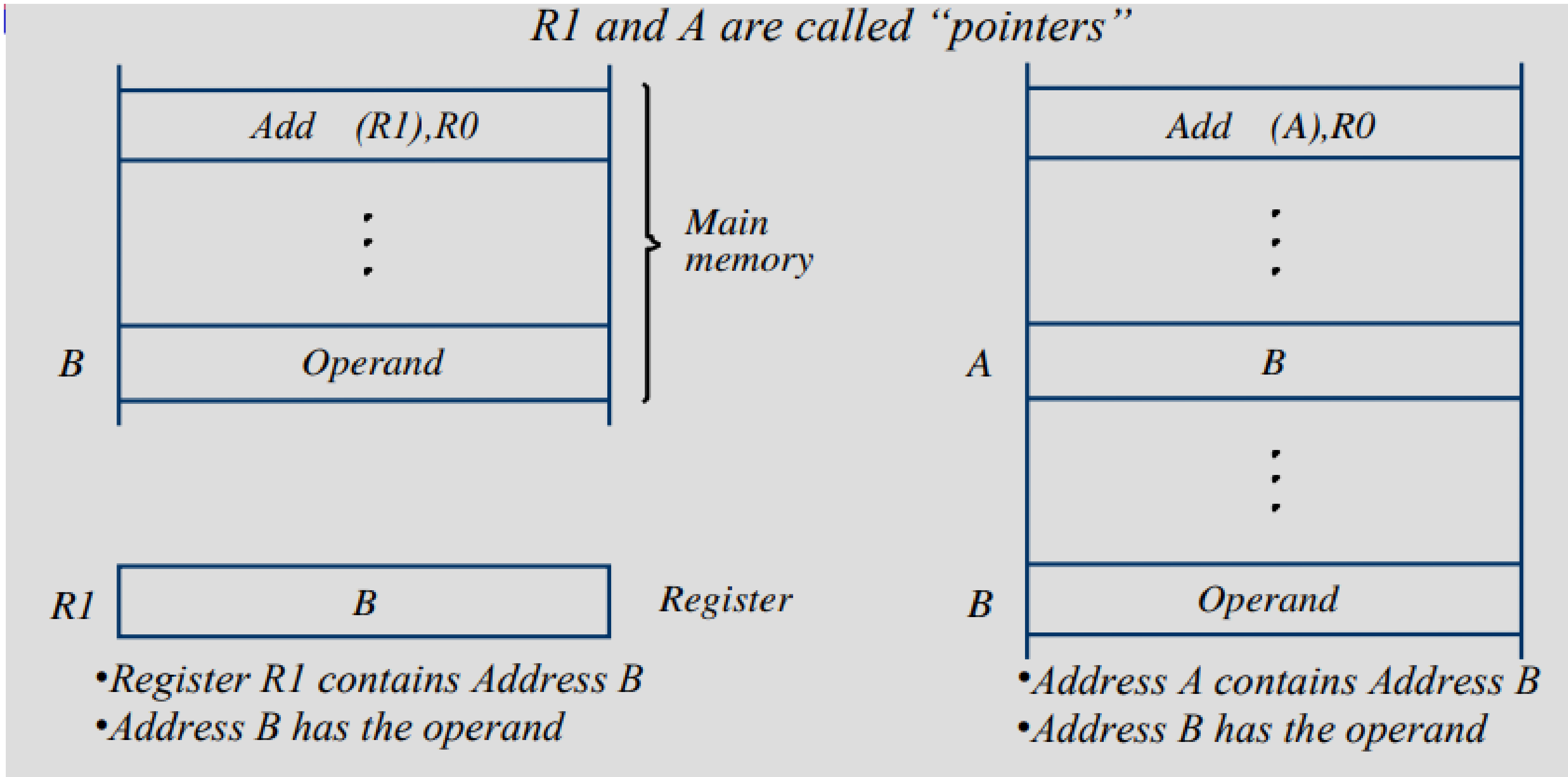
Continued…

- Register and absolute modes can be used to represent variables
  - 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.

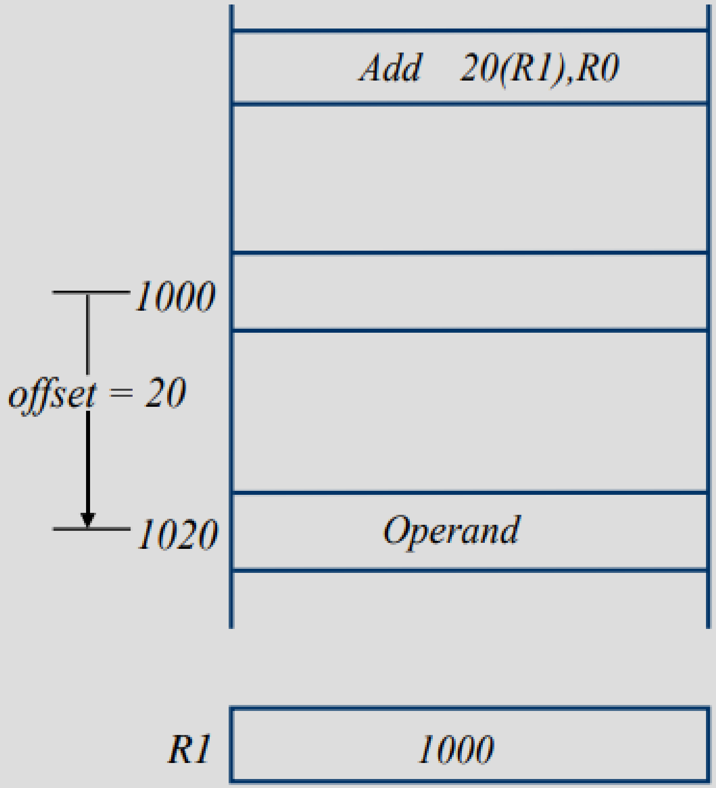
Indirect Mode:

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



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?

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"

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

Interrupt

- A suspension of a process such as the execution of a computer program, caused by an event external to that process, and performed in such a way that the process can be resumed.
- A way to improve processor utilization

Need For Interrupts?

- The OS is a reactive program
  - When you give some input
  - It will perform computations
  - Produces output BUT
  - Meanwhile you can interact with the system by interrupting the running process or
  - You can stop and start another process.
- This reactive ness is due to interrupts
- Modern Operating Systems Are Interrupt driven

Types of Interrupts

- There are three major types of interrupts that can cause a break in the normal execution of program
  - External Interrupts
  - Internal Interrupts
  - Software Interrupts

External Interrupts

- An external interrupt is a computer system interrupt that happens as a result of outside interference, whether that's
  - from the user,
  - from peripherals,
  - from other hardware devices or
  - through a network.
- These are different than internal interrupts that happen automatically as the machine reads through program instructions.
- Ex: I/O device requesting transfer of data, I/O device finished transfer of data, elapsed time of an event or power failure

Internal Interrupts

- An internal interrupt is a specific type of interrupt that is caused by
  - instructions embedded in the execution instructions of a program or
  - process.
- It resist changes by users, and happen "naturally" or "automatically" as a processor works through program instructions.
- Internal interrupts are called traps.
- Ex: register overflow, attempt to divide number by zero, invalid operation code, stack overflow

Software Interrupt

- A software interrupt is a type of interrupt that is caused either by a special instruction in the instruction set or by an exceptional condition in the processor itself.
- A software interrupt is invoked by software, unlike a hardware interrupt, and is considered one of the ways to communicate with the kernel or to invoke system calls, especially during error or exception handling.
- It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.
- Ex: switching of program from CPU mode to user mode

THANK YOU