Scheme of Solution

Internal Assessment Test 1 – September 2020

Sub:	COMPUTER ORGANIZATION							Code:	18CS34
Date:	12/09/2020	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	**ISE**
Answer Any **FIVE FULL** Questions									

Questionpaper and Scheme of Evaluation

Internal Assessment Test 1 – September 2020

Sub:	COMPUTER ORGANIZATION							Code:	18CS34
Date:	12/09/2020	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	**ISE**
Answer Any **FIVE FULL** Questions									

		Evaluation Marks	Total Marks
1(a)	Define the Little Endian and Big Endian assignments. Explain with relevant diagrams. - Explanation steps 3 Marks - Block Diagram 2 Marks	[3] [2]	[10]
1(b)	Explain the steps involved in instruction fetching from memory and execution in the processor with the help of block diagram. - Explanation steps 3 Marks - Block Diagram 2 Marks -	[3] [2]	
2(a)	Write an assembly level program to evaluate an arithmetic statement - Y= (A+B)*(C+D) using 0 address, 1 address, 2 address and 3 address instructions. - Explanation of each 2 marks - 0 address 2 Marks - 1 address 2 Marks - 2 address 2 Marks - 3 address 2 Marks	[2] [2] [2] [2] [2]	[10]

	-		
3(a)	Define Subroutine. How to pass parameters to subroutine? Explain with your own Example. - Explanation steps 3 Marks - Example 2 Marks - Parameter passing explanation 3Marks - Example 2 Marks	[3] [2] [3] [2]	[10]
4(a)	. What is Addressing mode? Explain any eight addressing modes with example of each mode. - Any 8addressing modes 8 Marks - Definition addressing modes with example 2 Marks	[5] [5]	[10]

5 (a)	What is performance measurement? Explain the overall SPEC rating of a computer. Calculate the SPEC rating for the program suite under test. Running times of the program suite for reference PC and PC under test are given below:			[5]	[10]
	Programs	Running time for reference PC	Running time for PC under test		
	P1	30	15		
	P2	100	50		
	P3	40	20		
	P4	10	5		
	P5	60	10		
(b)	-Performance measurement and equation-2 marks -problem 3 marks Explain the operation of stack with example. Write the line of code for implementing the same. Applications? - Explanation steps 2 Marks - Example 2 Marks - Application-1mark			[5]	
6 (a)	Write about shift and rotate instructions with neat diagram and example of each. - Shift instruction + example 5Marks - Rotate instruction + example 5 Marks			[5]	[10]

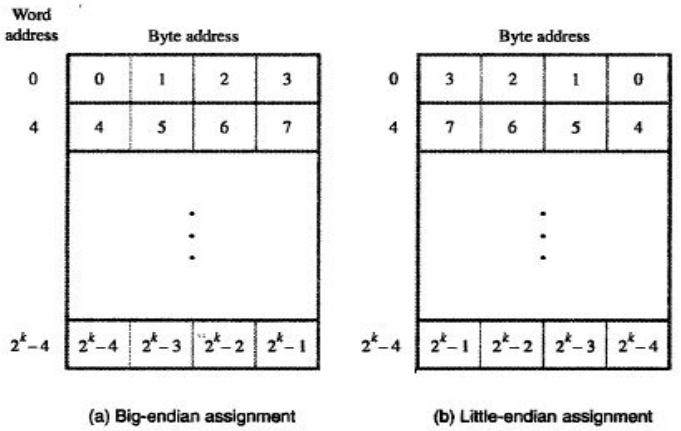
		[5]	
7 (a)	- With a neat diagram, explain I/O interface for an I/O devices. Explain various registers and flags involved in it. What are Condition Code flags? Explain the four commonly used flags. - Input and Output explanation 3 Marks - Block diagram 2 Marks -Four Code Flag + explanation 5 Marks	[3] [2] [3] [2]	[10]
8 (a)	What is assembler directive? List and explain any 4 assembler directives. What do you mean by instruction encoding? Definition-1 marks 4 assembler directive-6 Instruction encoding-3 marks	[1] [6] [3]	[1C0]

---

solution

1(a) (Define the Little Endian and Big Endian assignments. Explain with relevant diagrams

Two ways – Big Endian and Littler Endian
Big Endian – used when lower byte addresses used for MSB of the word
Little Endian – used when lower byte addresses use d for LSB of the word 2 Marks



(a) Big-endian assignment (b) Little-endian assignment

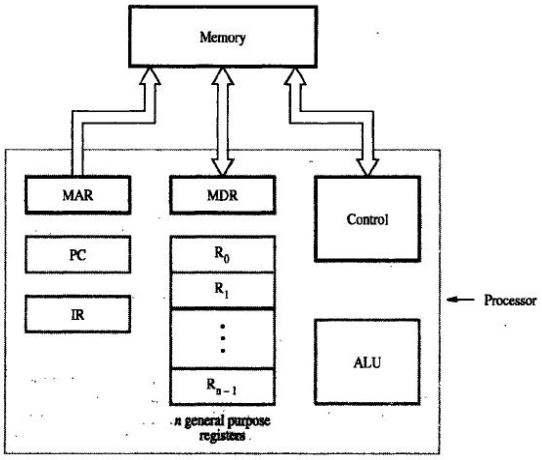
3 Marks

---

1 b) Explain the steps involved in instruction fetching from memory and execution in the processor with the help of block diagram. 5 Marks

• Transfer the contents of register PC to register MAR
• Issue a Read command to memory, and then wait until it has transferred the requested word into register MDR
• Transfer the instruction from MDR into IR and decode it
• Transfer the address LOCA from IR to MAR
• Issue a Read command and wait until MDR is loaded

• Content of the PC are incremented 3 Marks

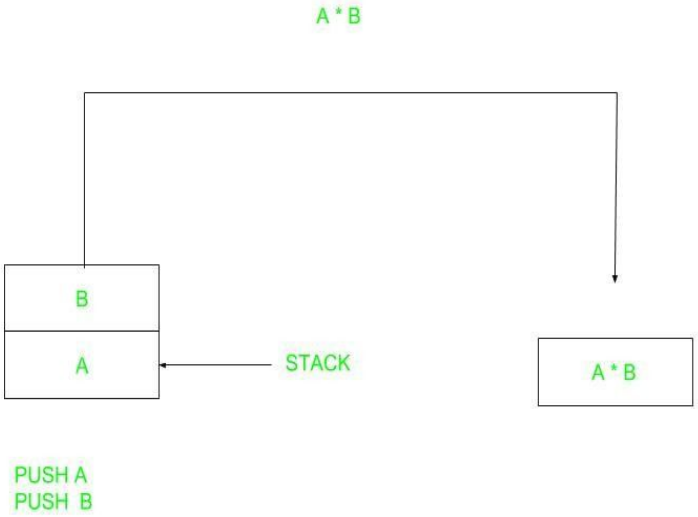


2 Marks

2(a) Write an assembly level program to evaluate an arithmetic statement

- Y= (A+B)*(C+D) using 0 address, 1 address, 2 address and 3 address instructions.

1. **Zero Address Instructions –**



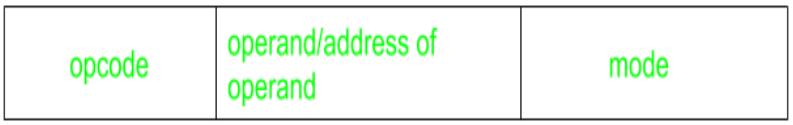
A stack based computer do not use address field in instruction.To evaluate a expression first it is converted to revere Polish Notation i.e. Post fix Notation.

```
Expression: X = (A+B)*(C+D)

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location
```

PUSH	A	TOP = A

PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

2. **One Address Instructions –**
   This use a implied ACCUMULATOR register for data manipulation.One operand is in accumulator and other is in register or memory location.Implied means that the CPU already know that one operand is in accumulator so there is no need to specify it.




```
Expression: X = (A+B)*(C+D)
AC is accumulator
M[] is any memory location
M[T] is temporary location
```

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

3. **Two Address Instructions –**
   This is common in commercial computers.Here two address can be specified in the instruction.Unlike earlier in one address instruction the result was stored in accumulator here result cab be stored at different location rather than just accumulator, but require more number of bit to represent address.

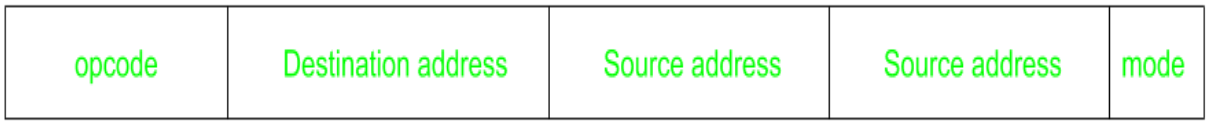
   Here destination address can also contain operand.

```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

MOV	R1, A	R1 = M[A]
ADD	R1, B	R1 = R1 + M[B]
MOV	R2, C	R2 = C
ADD	R2, D	R2 = R2 + D
MUL	R1, R2	R1 = R1 * R2
MOV	X, R1	M[X] = R1

4. **Three Address Instructions –**
   This has three address field to specify a register or a memory location. Program

created are much short in size but number of bits per instruction increase. These instructions make creation of program much easier but it does not mean that program will run much faster because now instruction only contain more information but each micro operation (changing content of register, loading address in address bus etc.) will be performed in one cycle only.




```
Expression: X = (A+B)*(C+D)
R1, R2 are registers
M[] is any memory location
```

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

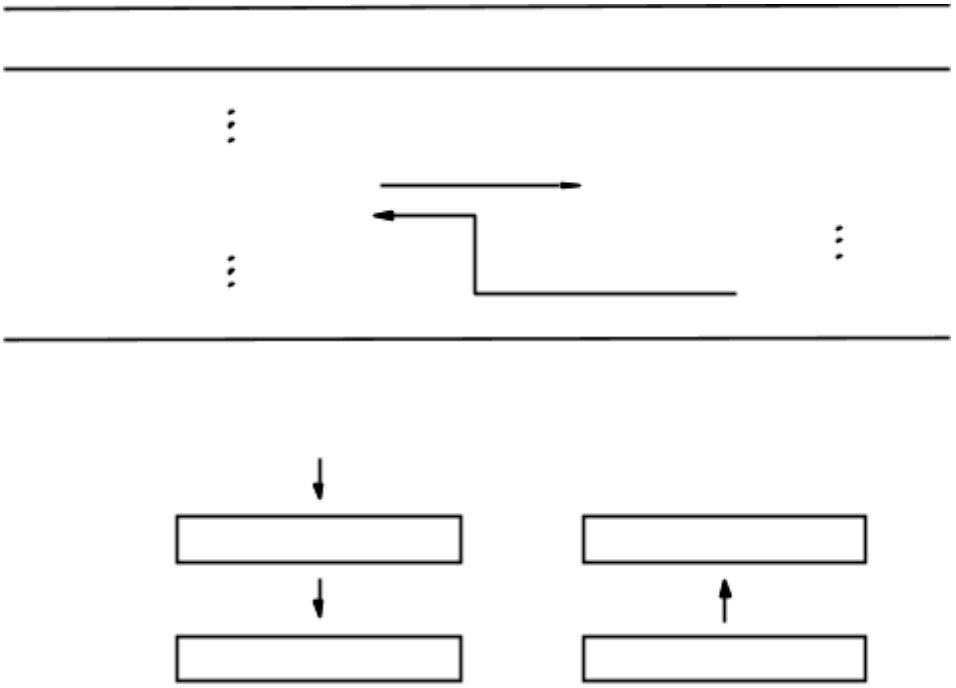
3(a) Define Subroutine. How to pass parameters to subroutine? Explain with Example. 10 Marks

In a program subtasks that are repeated on different data values are usually implemented as subroutines. When a program requires the use of a subroutine, it branches to the subroutine.
Branching to the subroutine is called as “calling” the subroutine.
Instruction that performs this branch operation is Call.
After a subroutine completes execution, the calling program continues with executing the instruction immediately after the instruction that called the subroutine.
Subroutine is said to “return” to the program.
Instruction that performs this is called Return.
Subroutine may be called from many places in the program. 3 Marks

Example:

- Calling program calls a subroutine, whose first instruction is at address 1000.
- The Call instruction is at address 200.
- While the Call instruction is being executed, the PC points to the next instruction at address 204.
- Call instructions stores address 204 in the Link register, and loads 1000 into the PC.
- Return instruction loads back the address 204 from the link register into the PC.

2 Marks



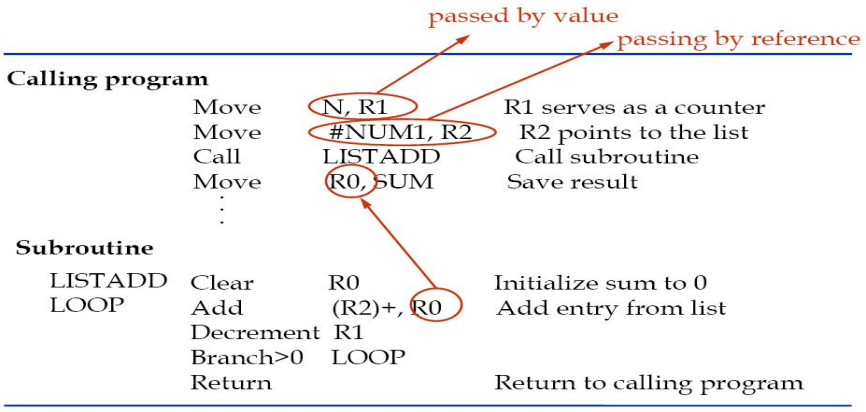
Parameter Passing:

When calling a subroutine, a program must provide to the subroutine the **parameters**, that is, the **operands** or their addresses, to be used in the computation. Later, the subroutine **returns** other parameters, in this case, the **result of computation. T**he exchange of information between a calling program and a subroutine is referred to as parameter passing

Parameter passing approaches

- The parameters may be placed in registers or in memory locations, where they can be accessed by the subroutine
- The parameters may be placed on the processor stack used for saving the return address

3 Marks



2 Marks

4(a) Explain any five addressing modes with example of each mode. 10 Marks

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand=Value
Register	Ri	EA=Ri
Absolute (Direct)	LOC	EA=LOC
Indirect	(Ri)	EA=[Ri]
	(LOC)	EA=[LOC]
Index	X(Ri)	EA=[Ri]+X
Base with index	(Ri, Rj)	EA=[Ri]+[Rj]
Base with index and offset	X(Ri, Rj)	EA=[Ri]+[Rj]+X
Relative	X(PC)	EA=[PC]+X
Autoincrement	(Ri)+	EA=[Ri]; Increment Ri
Autodecrement	-(Ri)	Decrement Ri; EA=[Ri]

Any five addressing modes 5 Marks
Example of each addressing modes 5 Marks

5(a) Calculate the SPEC rating for the program suite under test. Running times of the program suite for reference PC and PC under test are given below:
5 Marks

Programs	Running time for reference PC	Running time for PC under test
P1	10	20
P2	100	50
P3	40	20
P4	10	5
P5	60	30

P1=10/20= 0.5 1 mark

P2= 100/50=2 1 mark

P3=40/20=2 1 mark

P4=10/5=2 1 mark

P5=60/30=2 1 mark

Overall SPEC rating= $(0.5 \times 2 \times 2 \times 2 \times 2)^{1/5} = 1.517$ 5 marks

---

5(b) Explain the operation of stack with example. Write the line of code for implementing the same. 5 Marks

A stack is a list of data elements, usually words or bytes with the accessing restriction that elements can be added or removed at one end of the stack. End from which elements are added and removed is called the "top" of the stack. Other end is called the "bottom" of the stack. Also known as: Push down stack. Last in first out (LIFO) stack.*Push* - placing a new item onto the stack. *Pop* - Removing the top item from the stack. 3 Marks

Example:

- Processor with 65536 bytes of memory.
- Byte addressable memory.
- Word length is 4 bytes.
- First element of the stack is at BOTTOM.
- SP points to the element at the top.
- Push operation can be implemented as:

Subtract #4, SP
Move A, (SP)

- Pop operation can be implemented as:

Move (SP), B
Add #4, SP

- Push with autodecrement:

Move A, -(SP)

- Pop with autoincrement:

Move (SP)+, A 2 Marks

---

6 (a).What is an interrupt? List the sequence of events involved in handling an interrupt request from a single device. 5 Marks

An approach for the I/O device to alert the processor when it becomes ready. It is done by sending a hardware signal called an interrupt to the processor. At least one of the bus control lines, called an interrupt-request line is dedicated for this purpose. 2 Marks

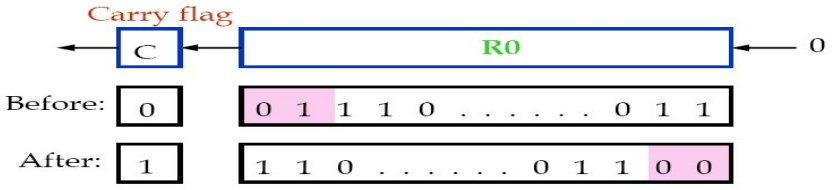
The device raises an interrupt request
The processor interrupts the program currently being executed
Interrupts are disabled by changing the control bits in the Processor Status (PS) register
The device is informed that its request has been recognized, and in response, it deactivates the interrupt request signal
The action requested by the interrupt is performed by the interrupt-service routine
Interrupts are enabled and execution of the interrupted program is resumed 3 Marks

6 (b) Write about shift and rotate instruction with neat diagram and example of each. 5 Marks

- Logical shifts

Logic shift left

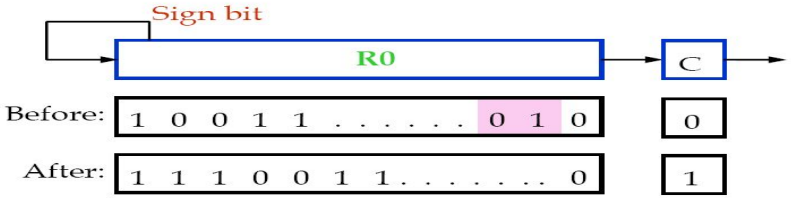
LShiftL #2, R0



- Arithmetic shifts

shift right

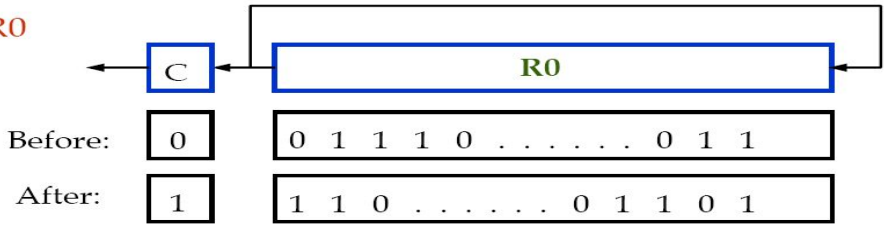
AShiftR #2, R0



3 Marks

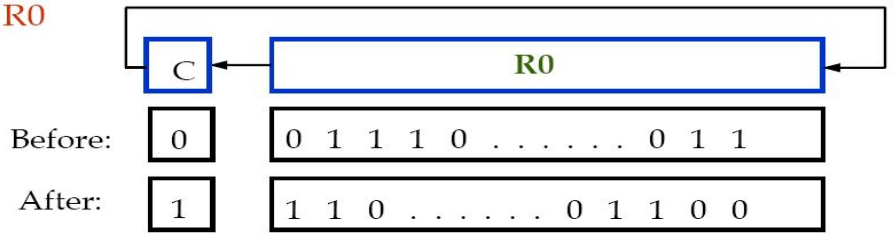
- Rotate left without carry

RotateL #2, R0



- Rotate left with carry

RotateLC #2, R0



2 Marks

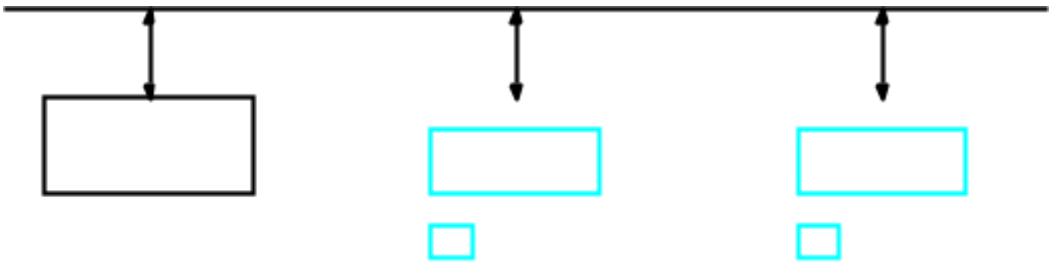
7(a) With a neat diagram, explain I/O interface for an I/O device. Also, explain various registers involved in it. 5 Marks

Input:

- When a key is struck on the keyboard, an 8-bit character code is stored in the buffer register DATAIN.
- A status control flag SIN is set to 1 to indicate that a valid character is in DATAIN.
- A program monitors SIN, and when SIN is set to 1, it reads the contents of DATAIN.
- When the character is transferred to the processor, SIN is automatically cleared.
- Initial state of SIN is 0.

Output:

- When SOUT is equal to 1, the display is ready to receive a character.
- A program monitors SOUT, and when SOUT is set to 1, the processor transfers a character code to the buffer DATAOUT.
- Transfer of a character code to DATAOUT clears SOUT to 0.
- Initial state of SOUT is 1. 3 Marks



2 Marks

What are Condition Code flags? Explain the four commonly used flags. 5 Marks

- The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions.
- This is accomplished by recording required information in individual bits, often called condition code flags
- Four commonly used flags are
  - N (negative): set to 1 if the results is negative; otherwise, cleared to 0
  - Z (zero): set to 1 if the result is 0; otherwise, cleared to 0
  - V (overflow): set to 1 if arithmetic overflow occurs; otherwise, cleared to 0
  - C (carry): set to 1 if a carry-out results from the operation otherwise, cleared to 0
- N and Z flags caused by an arithmetic or a logic operation,
- V and C flags caused by an arithmetic operation 5 Marks

---

8. What is assembler directive? List and explain any 4 assembler directives. What do you mean by instruction encoding?

Assembler directives- These are the statements that direct the assembler to do something. As the name says, it directs the assembler to do a task.

The specialty of these statements is that they are effective only during the assembly of a program but they do not generate any code that is machine executable.

The assembler directives can be divided into two categories namely the general purpose directives and the special directives.

They are classified into the following categories based on the function performed by them- Simplified segment directives Data allocation directives Segment directives Macros related directives Code label directives Scope directives Listing control directives Miscellaneous directives

.CODE- This assembler directive indicates the beginning of the code segment. Its format is as follows: .CODE [name] The name in this format is optional.

.DATA- This directive indicates the beginning of the data segment.

.MODEL- This directive is used for selecting a standard memory model for the assembly language program. Each memory model has various limitations depending on the maximum space available for code and data.

.STACK- This directive is used for defining the stack. Its format is as follows: .STACK [size]

Define Byte [DB]- This directive defines the byte type variable.

Define Word [DW]- The DW directive defines items that are one word (two bytes) in length.

Define Double word [DD]- It defines the data items that are a double word (four bytes) in length.

Define Quad word [DQ]- This directive is used to tell the assembler to declare variable 4 words in length or to reserve 4 words of storage in memory.

Define Ten bytes [DT]- It is used to define the data items that are 10 bytes long.

ASSUME- The directive is used for telling the assembler the name of the logical segment which should be used.

END- This is placed at the end of a source and it acts as the last statement of a program. This is because the END directive terminates the entire program.

ALIGN- This directive will tell the assembler to align the next instruction on an address which corresponds to the given value.

LABEL- This directive assigns name to the current value of the location counter.

INCLUDE- This directive is used to tell the assembler to insert a block of source code from the named file into the current source module. This shortens the source code. Its format is:

Instructions consist of:

- **operation** (opcode) e.g. MOV
- **operands** (number depends on operation)
- operands specified using addressing modes
- addressing mode may include **addressing information**
- e.g. registers, constant values

Encoding of instruction must include opcode, operands & addressing information.

Encoding:

- represent entire instruction as a **binary value**
- **number of bytes** needed depends on how much information must be encoded
- instructions are **encoded by assembler:**
- **.OBJ** file ! (link, then loaded by loader)
- instructions are decoded by processor during execution cycle

We will consider a subset of interesting cases

Instructions with No Operands (easy)

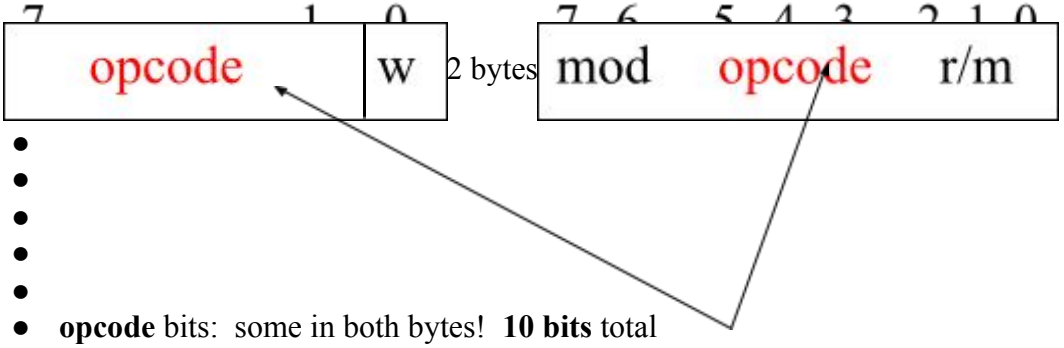
- encode operation only in a single byte
- examples:

RET **C3** H NOP **90** H

- Are consistent – never change

Instructions with One Operand

- operand is a register (reg8/16) or a memory operand (mem8/16)
- always 2 bytes for opcode and addressing info
- may have up to 2 more bytes of immediate data



- **opcode** bits: some in both bytes! **10 bits** total
- **w** = width of operand

0 = 8-bit

1 = 16-bit

INCLUDE path: file name