Lecture 4

LOGIC GATES AND DATA MANIPULATION

Data Storage

Bits and Their Storage

Main Memory

Mass Storage

Representing Information as Bit Patterns

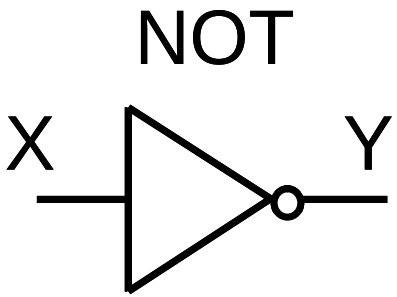
Storing Numbers

Gates

Gate: A device that computes a Boolean operation

- Often implemented as (small) electronic circuits
- Provide the building blocks from which computers are constructed

NOT Gate -- Inverter



Y = ~X

X	Y
0	1
1	0

NOT

Y = ~X

Y = !X

Y = not X

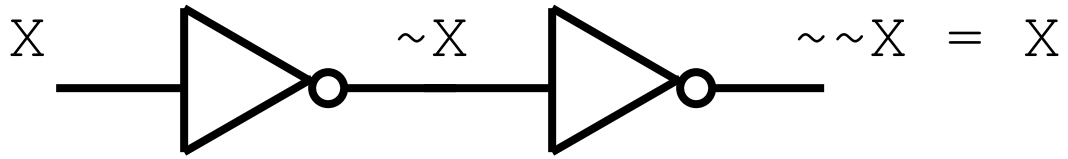
Y = X’

Y = ¬ X

Y = — X

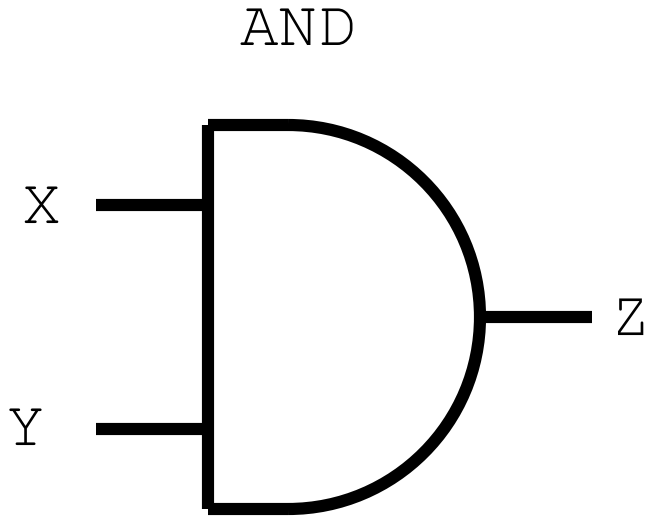
not(Y,X)

NOT



X	~X	~~X
0	1	0
1	0	1

AND Gate



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Z = X & Y

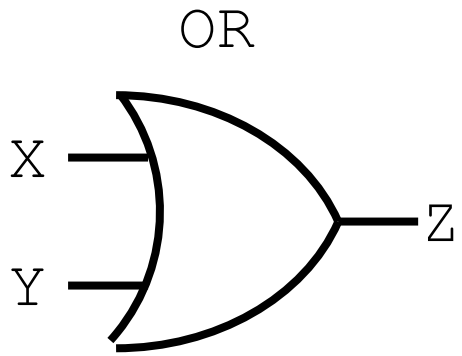
AND

X & Y

X and Y

and(Z,X,Y)

OR Gate



Z = X | Y

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

OR

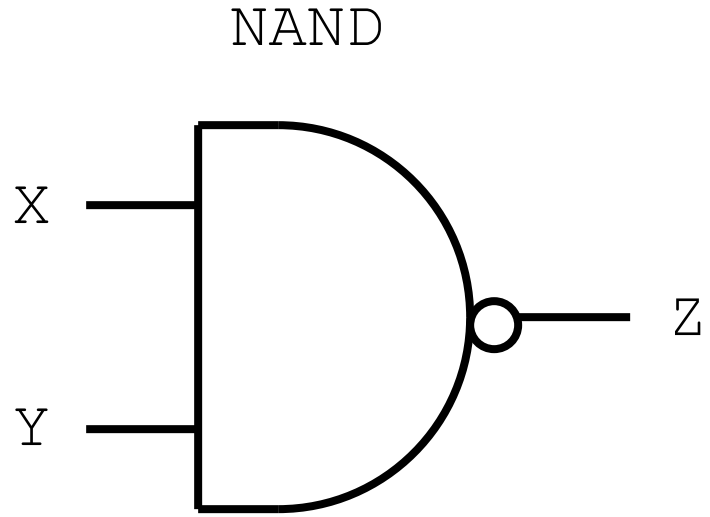
X | Y

X or Y

X + Y

or(Z,X,Y)

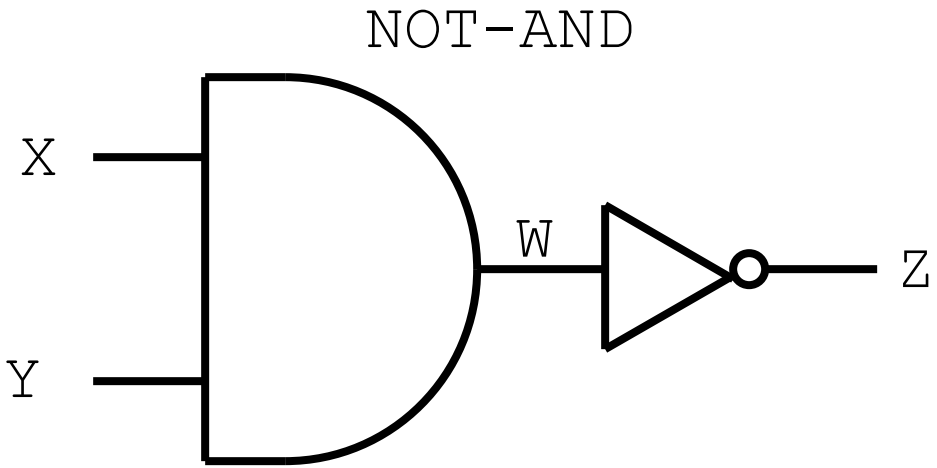
NAND Gate



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

```
Z = ~(X & Y)
nand(Z,X,Y)
```

NAND Gate

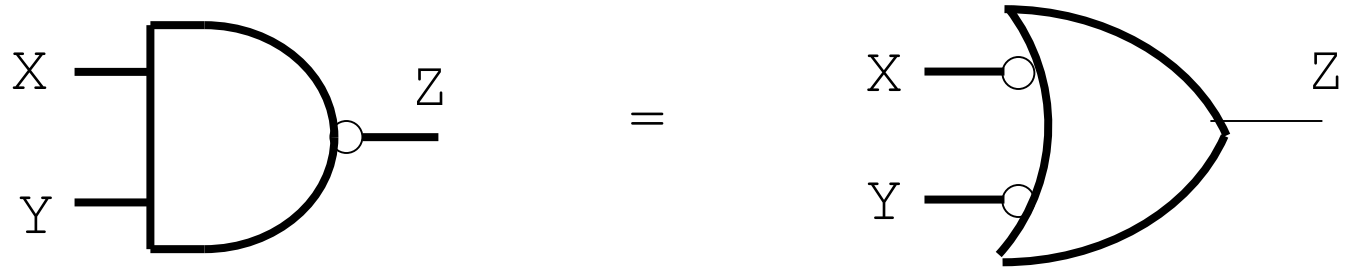


X	Y	W	Z
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

W = X & Y

Z = ~W = ~(X & Y)

NAND Gate



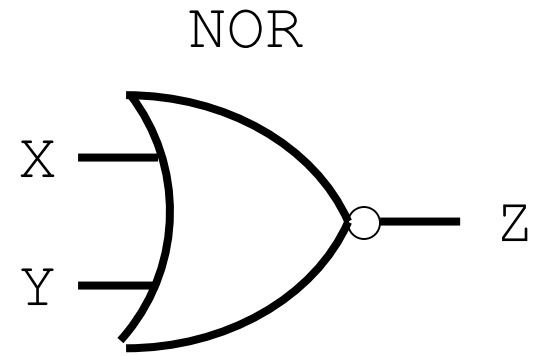
Z = ~(X & Y)

X	Y	W	Z
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

Z = ~X | ~Y

X	Y	~X	~Y	Z
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

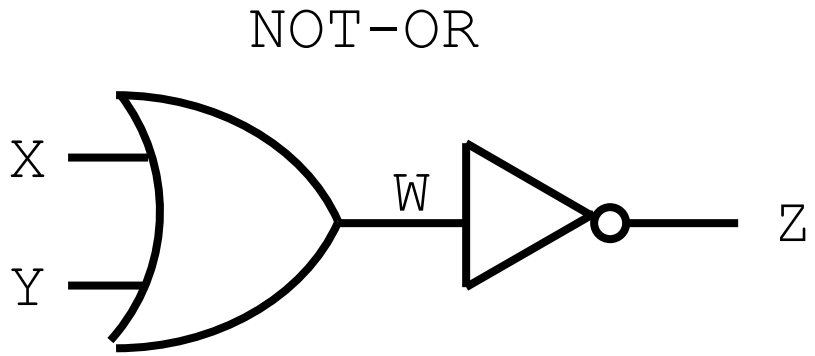
NOR Gate



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

```
Z = ~(X | Y)
nor(Z,X,Y)
```

NOR Gate

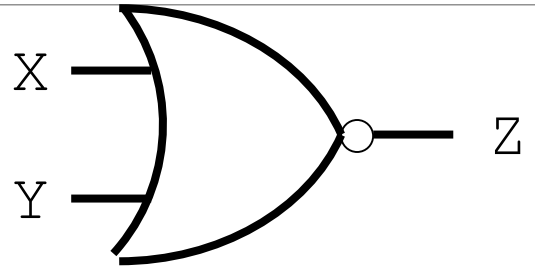


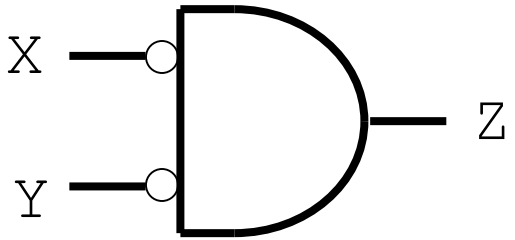
X	Y	W	Z
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

```
W = X | Y

Z = ~W = ~(X | Y)
```

NOR Gate





`Z = ~(X | Y)`

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

`Z = ~X & ~Y`

X	Y	~X	~Y	Z
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

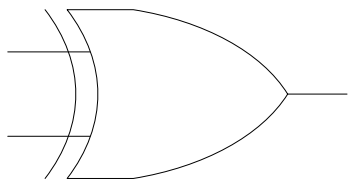
Exclusive-OR Gate

XOR

X

Y

Z



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

```
Z = X ^ Y
xor(Z,X,Y)
```

$X \oplus Y$

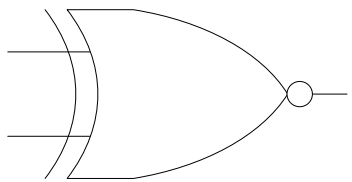
Exclusive-NOR Gate

XNOR

X

Y

Z



```
Z = ~(X ^ Y)
Z = X ~^ Y
xnor(Z,X,Y)
```

X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

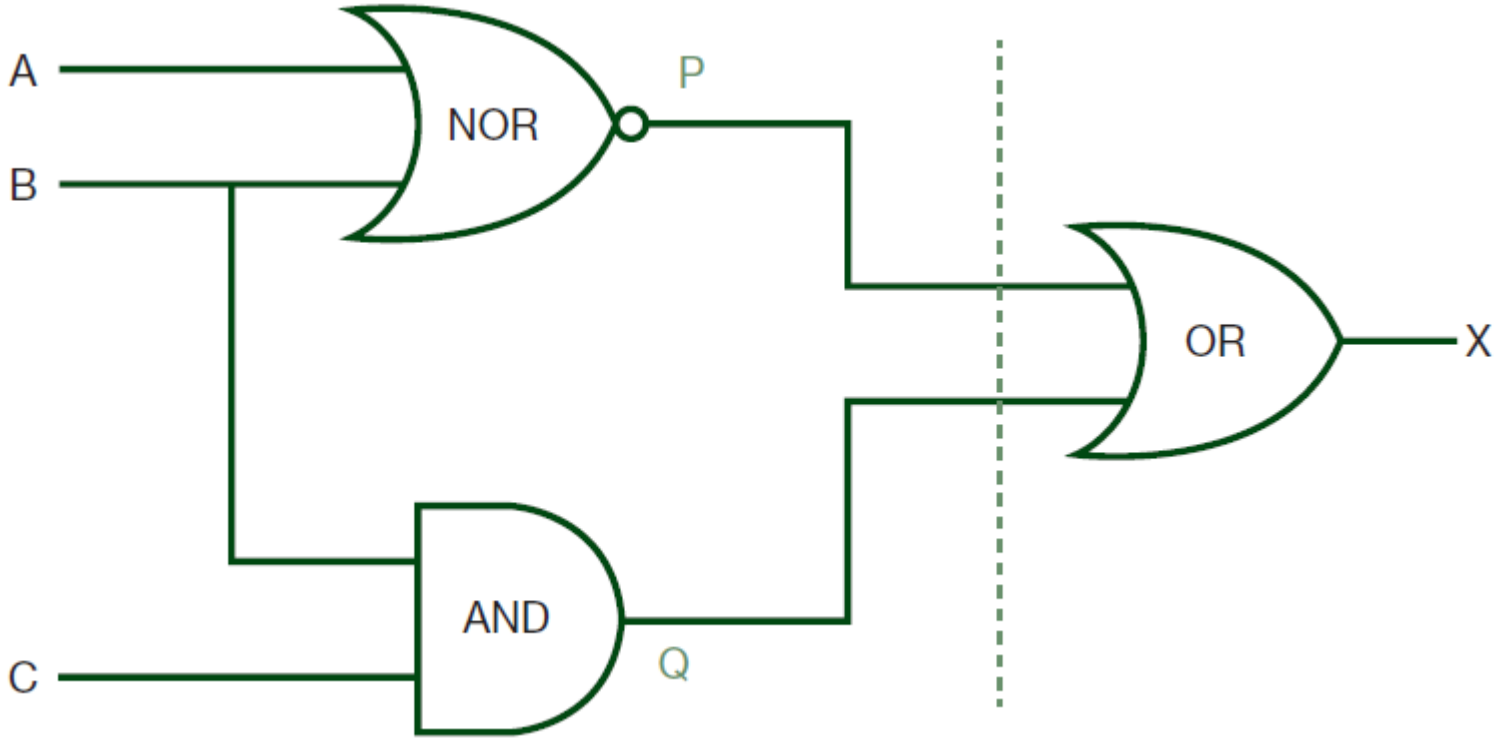
Digital logic operations examples

Perform the indicated operations

- 01001011 AND 10101011
  - Solution: 00001011
- 01001011 OR 10101011
  - Solution: 11101011
- 01001011 XOR 10101011
  - Solution: 11100000

Digital logic operations examples

Example: **Produce a truth table from the following logic circuit**



Digital logic operations examples

Answer

- First part
- There are 3 inputs; thus we must have (i.e. 8) possible combinations of 1s and 0s.
- To find the values (outputs) at points P and Q, it is necessary to consider the truth tables for the NOR gate (output P) and the AND gate (output Q) i.e.:
  - P = A NOR B
  - Q = B AND C

Digital logic operations examples

We thus get

P = A **NOR B**
Q = B **AND C**

INPUT A	INPUT B	INPUT C	OUTPUT P	OUTPUT Q
0	0	0	*1*	*0*
0	0	1	*1*	*0*
0	1	0	*0*	*0*
0	1	1	*0*	*1*
1	0	0	*0*	*0*
1	0	1	*0*	*0*
1	1	0	*0*	*0*
1	1	1	*0*	*1*

Digital logic operations examples

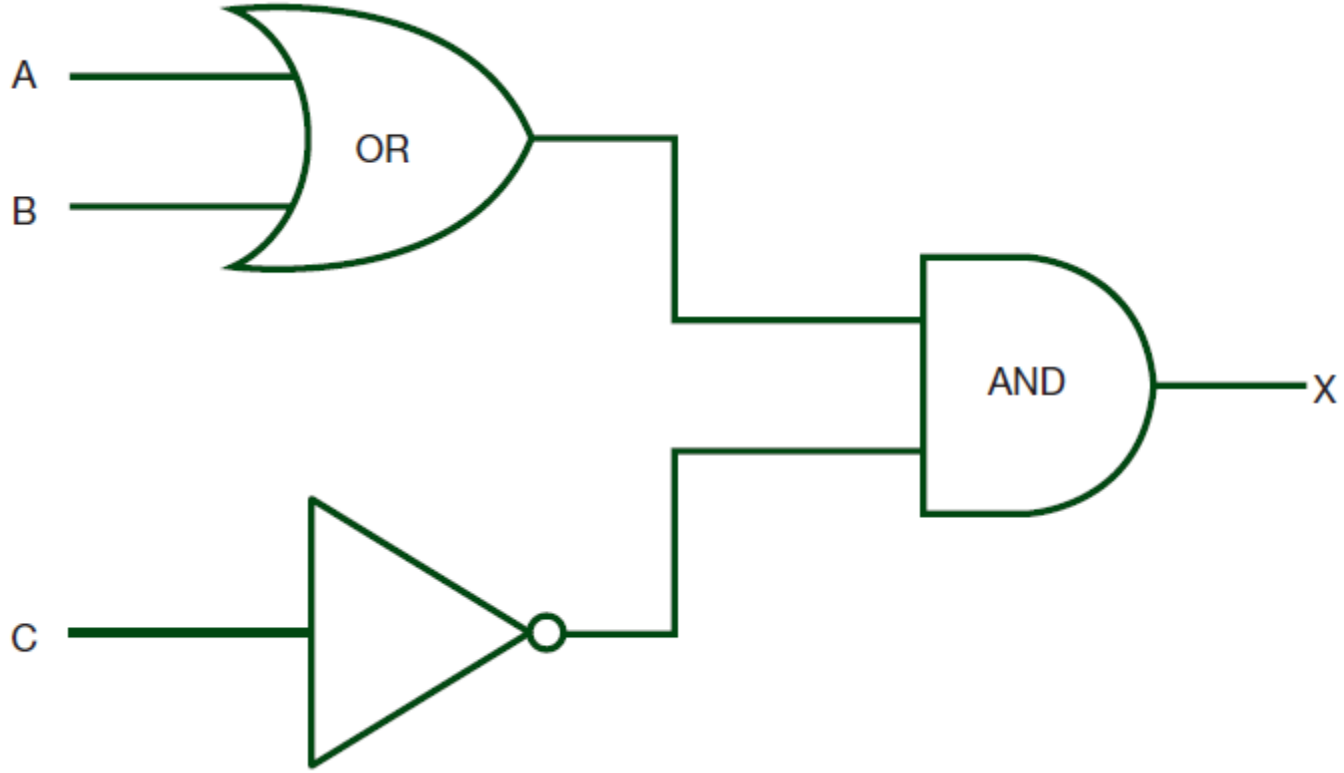
Second part

- There are 8 values from P and Q which form the inputs to the last OR gate.
- Hence we get X = P OR Q

INPUT P	INPUT Q	OUTPUT X
1	0	1
1	0	1
0	0	0
0	1	1
0	0	0
0	0	0
0	0	0
0	1	1

Digital logic operations examples

Example: : Produce a truth table from the following logic circuit



Digital logic operations examples

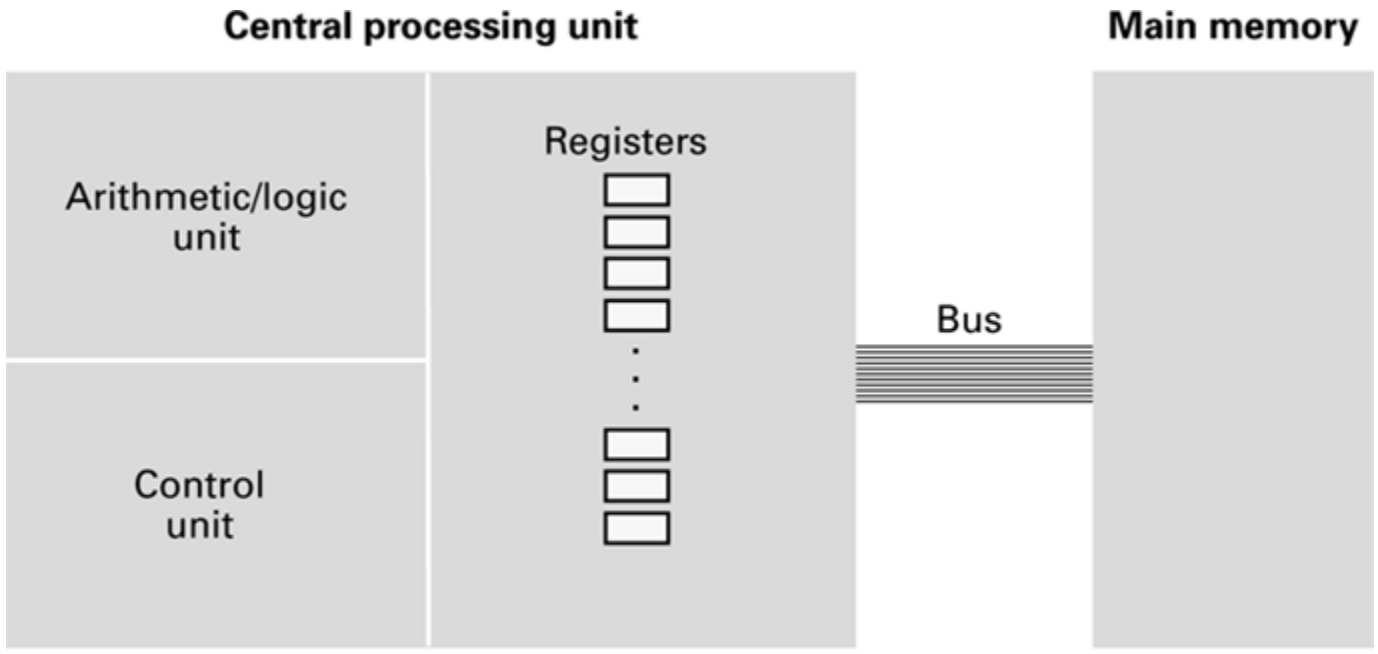
Answer x=(A OR B) AND (NOT C)

INPUT A	INPUT B	INPUT C	OUTPUT X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Computer Architecture

- Central Processing Unit (CPU) or processor
  - Arithmetic/Logic unit
  - Control unit
  - Registers
- Bus
- Motherboard

CPU and main memory connected via a bus



Registers

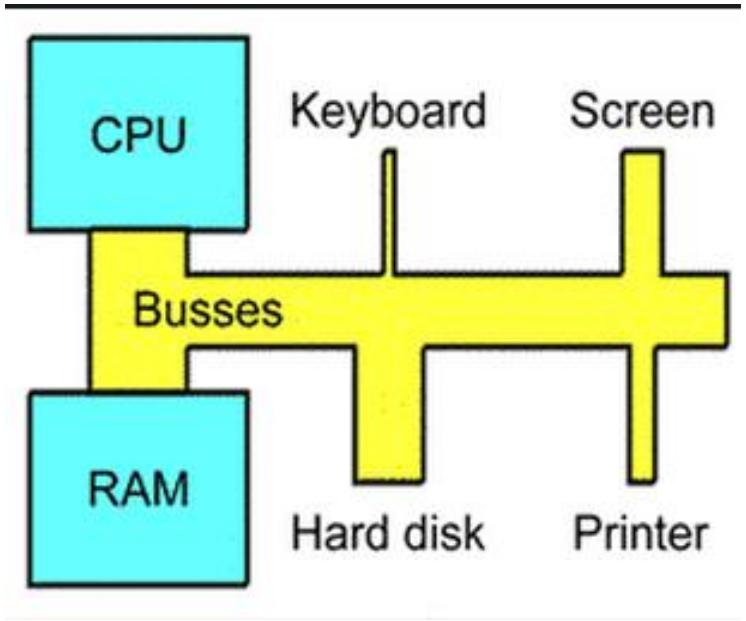
Difference between a CPU's cache and register

- A CPU cache is a cache used by the central processing unit(CPU) of a computer to reduce the average time to access data from the main memory.

- The Registers are rather like variables in a high level language and are used to store values which are actually being processed. For example with C = A + B, the values of A and B are loaded into Registers (like assigning variables) and then the instruction 'add A and B and store in C' is executed.

Motherboard

A motherboard (sometimes alternatively known as the mainboard, system board, planar board or logic board). It holds many of the crucial electronic components of the system, such as the central processing unit(CPU) and memory, and provides connectors for other peripherals.



Stored Program Concept

A program can be encoded as bit patterns and stored in main memory. From there, the CPU can then extract the instructions and execute them. In turn, the program to be executed can be altered easily.

Terminology

- **Machine instruction:** An instruction (or command) encoded as a bit pattern recognizable by the CPU
- **Machine language:** The set of all instructions recognized by a machine

Adding values stored in memory

Step 1. Get one of the values to be added from memory and place it in a register.

Step 2. Get the other value to be added from memory and place it in another register.

Step 3. Activate the addition circuitry with the registers used in Steps 1 and 2 as inputs and another register designated to hold the result.

Step 4. Store the result in memory.

Step 5. Stop.

Dividing values stored in memory

Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

Step 3. If this second value is zero, JUMP to Step 6.

Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

Step 6. STOP.

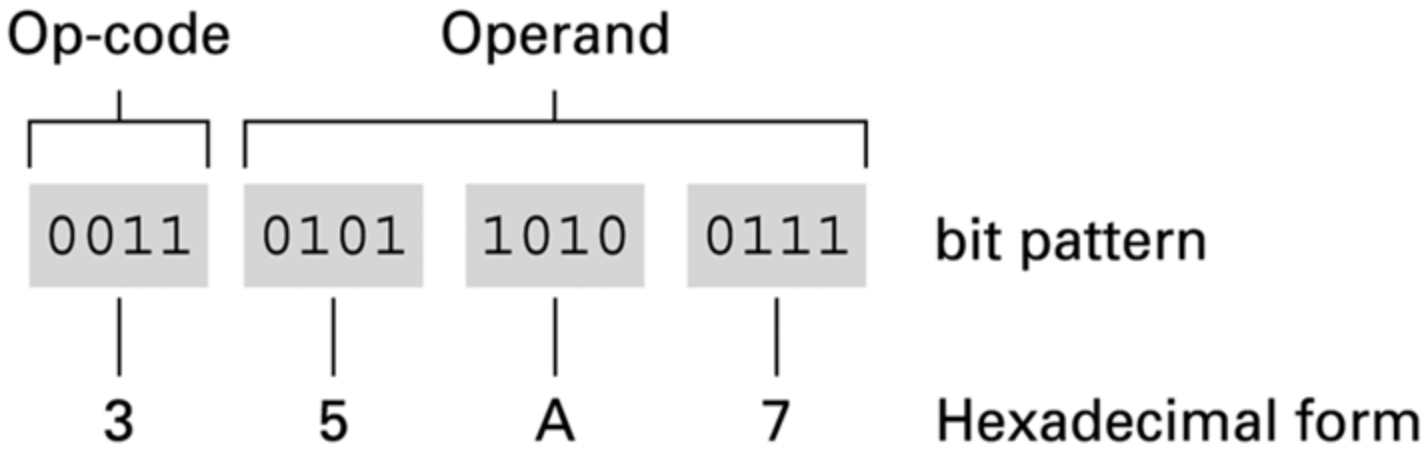
Parts of a Machine Instruction

Op-code: Specifies which operation to execute

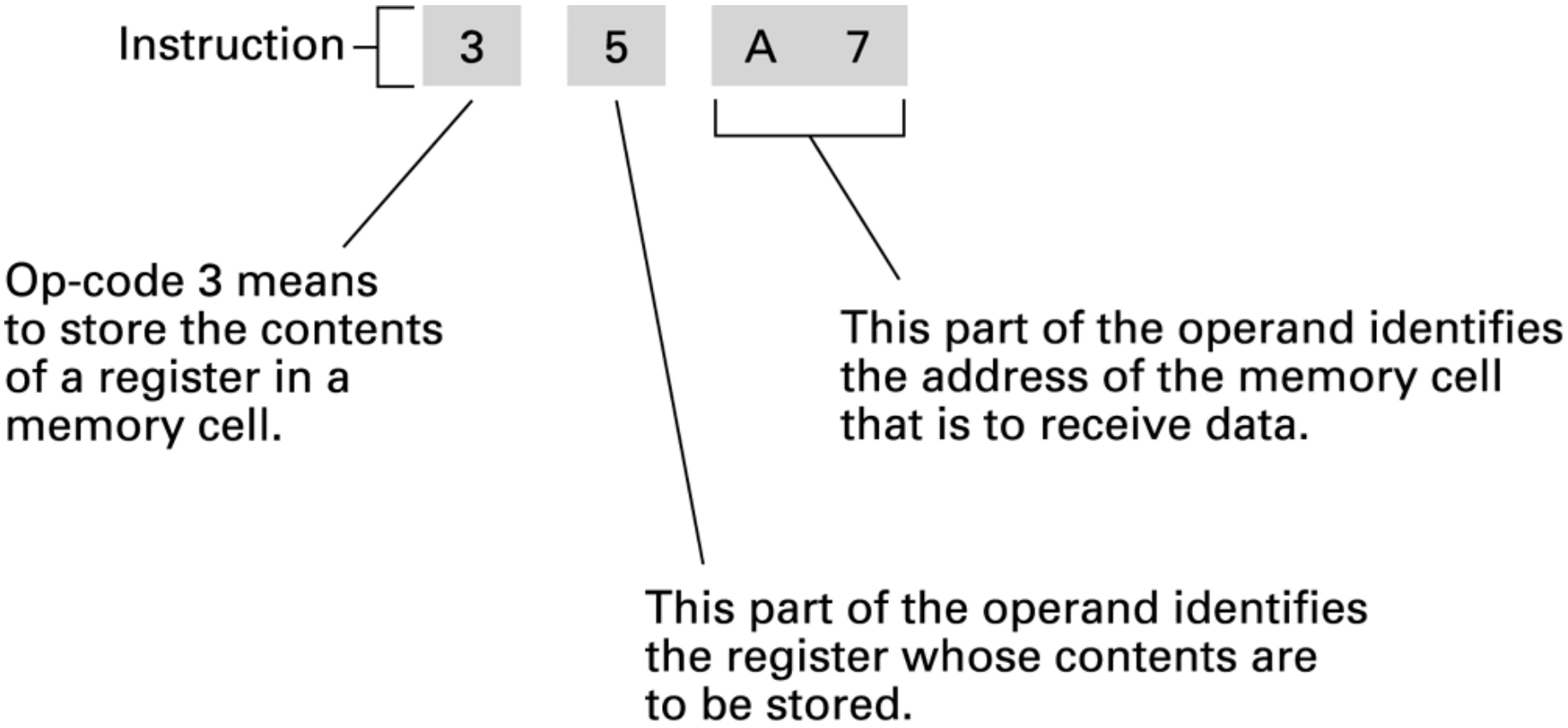
Operand: Gives more detailed information about the operation

- Interpretation of operand varies depending on op-code

The composition of an instruction for the machine in Appendix C



Decoding the instruction 35A7



An encoded version of the instructions

Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

Appendix C example

OpCode	Operand	Description
1	RXY	LOAD the register R with the bit pattern found in memory cell whose address is XY. E.g., 14A3 would cause the contents of memory cell at address A3 to be placed in register 4.
2	RXY	IMMEDIATE LOAD the register R with the bit pattern XY. E.g., 20A3 would cause the value A3 to be placed in register 0.
3	RXY	STORE the bit pattern found in register R in the memory cell whose address is XY. E.g., 35B1 would cause the contents of register 5 to be placed in the memory cell whose address is B1.
4	0RS	MOVE the bit pattern found in register R to register S. Note that not all bits in the operand are used. E.g., 40A4 would cause the contents of register A to be copied to register 4.

Appendix C

5	RST	ADD the bit patterns in register S and T as though they were two's complement numbers and store the result in register R. E.g. 5726 would cause the values in registers 2 and 6 to be added and placed in register 7.
6	RST	ADD the bit patterns in register S and T as though they were floating point numbers and store the result in register R. E.g. 6726 would cause the values in registers 2 and 6 to be added as floating point numbers and placed in register 7.
7	RST	LOGICALLY OR the bit patterns in registers S and T and place the result in register R. E.g., 7CB4 would cause the result of ORing the contents of registers B and 4 to be placed in register C.
8	RST	LOGICALLY AND the bit patterns in registers S and T and place the result in register R. E.g., 8045 would cause the result of ANDing the contents of registers 4 and 5 to be placed in register 0.
9	RST	EXCLUSIVE OR the bit patterns in registers S and T and place the result in register R. E.g., 95F3 would cause the result of EXCLUSIVE Oring the contents of registers F and 3 to be placed in register 5.
A	R0X	ROTATE the bit pattern in register R one bit to the right X times. Each time place the bit that started at the low-order end to the high-order end. E.g., A403 would cause the contents of register 4 to be rotated 3 bits to the right in a circular fashion.
B	RXY	JUMP to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register 0. Otherwise, continue with the normal sequence of execution. E.g., B43C would first compare the contents of register 4 with register 0. If identical, the execution sequence would be altered so the next instruction executed is the one at location 3C in memory. Otherwise, program execution continues as normal with the next sequential instruction.
C	000	HALT instruction, stop execution. E.g. C000 stops execution.

ANNOUNCEMENTS

❖ The **Lecture 5** were posted Online Facebook Group last week.

Please read them carefully.

❖**Sheet # 4** were posted online this week.

❖**Submissions of Sheet #4** is during next week's Labs

❖**Quiz # 3** will be held in the week after(Data Storage & Logic Gates)

Thank you

References

Computer Science: An Overview -Eleventh Edition –by J. Glenn Brookshear

Computer Fundmental –Pradeep K. Sinha & Priti Sinha

Foundations of Computer Science @ Cengage Learning