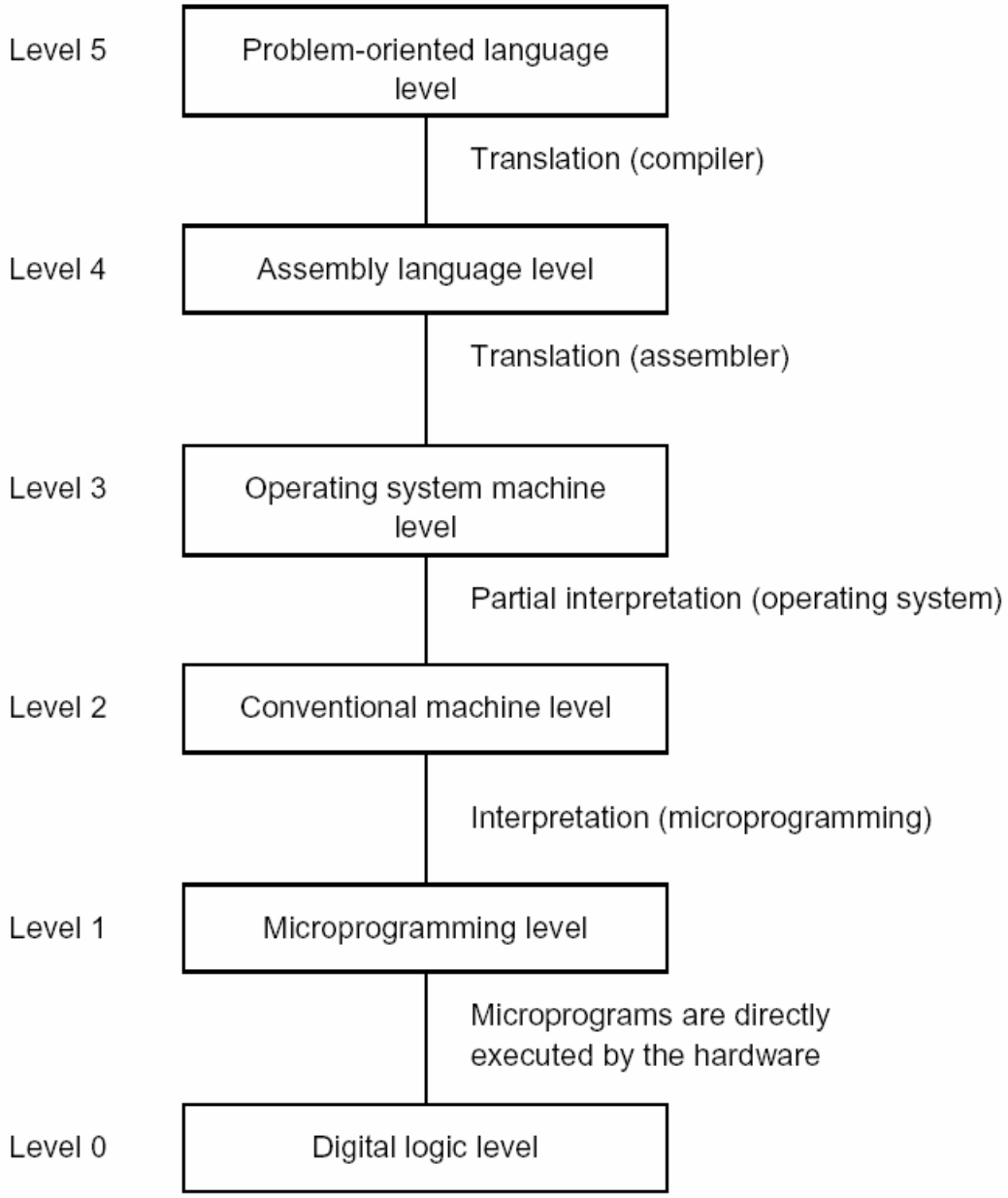
Multilevel Machines



The Digital Logic Level

Operations at the digital logic level are performed by circuits called logic gates.

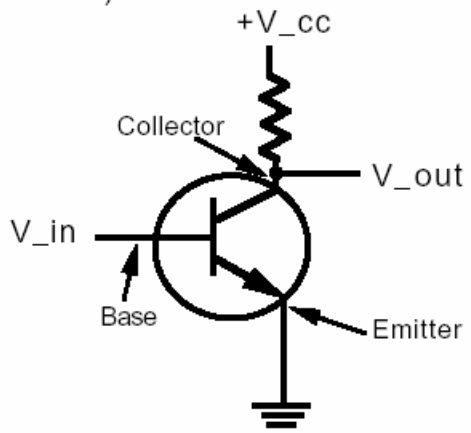
Each gate has one or more digital inputs (signals representing 0 or 1) and computes as output some simple function of these inputs, such as AND, OR, Exclusive OR, etc.

Each gate is built up from a small number of transistors. There are other levels below level 0 that describe, for example, the manner in which the individual devices such as transistors and diodes are used in constructing the logic gates or that describe the solid state physics behind the operation of the individual devices.

In this course we will ignore the device and solid state physics levels and only briefly touch on the digital logic level. The lower levels are more appropriate for a course in computer hardware design. We will concentrate on levels 1 and above.

The Digital Logic Level

The transistors with digital devices act basically as high speed binary switches. Their operation can be described in a very simplified form as follows: each contains three external connections: the collector, the base and the emitter. When the input voltage V_in is below a certain critical value, the transistor turns off and acts like an infinite resistance, causing the output, V_out, to take on a value close to V_cc. V_cc is an externally regulated voltage which is typically +5 volts. When V_in exceeds the critical value, the transistor switches on and acts like a wire, causing V_out to be pulled down to ground (by convention 0 volts).



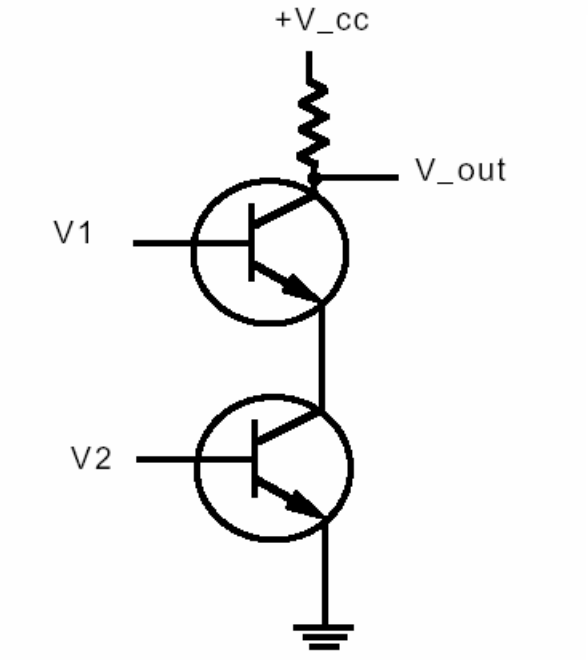
V_out is the inverse of V_in. This circuit consitutes an inverter or NOT gate.

The Digital Logic Level

Typically, a signal between 0 and 1 volt represents binary 0 (i.e. a low value) and a signal between 2 and 5 volts represents a binary 1 (or high value) in digital circuits that employ positive logic. Negative logic corresponds to the opposite relationship in which a low voltage means binary 1 and the higher voltage means a binary 0.

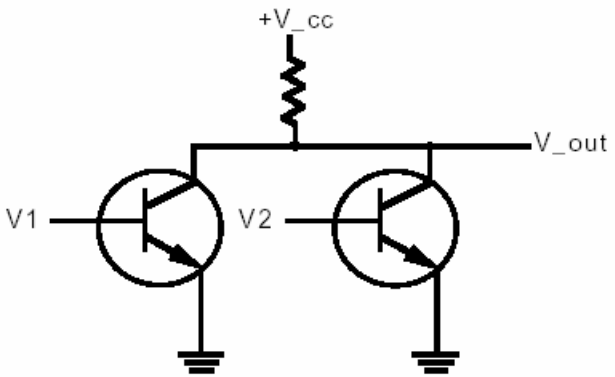
The time required for a transistor to switch from one state to the other is on the order of a few nanoseconds.

Shown below are two transistors cascaded in series. If both V1 and V2 are high, both transistors will conduct and V_out will be pulled low. If either input is low, the corresponding transistor will turn off, and the output will be high. In other words, V_out is low if and only if both V1 and V2 are high. Thus this circuit constitutes what is known as a NAND gate.



The Digital Logic Level

By wiring the two transistors in parallel rather than in series, we obtain a circuit that corresponds to a NOR gate in which the output is high only if neither of the inputs is high. That is, the ouput is high when both inputs are low.



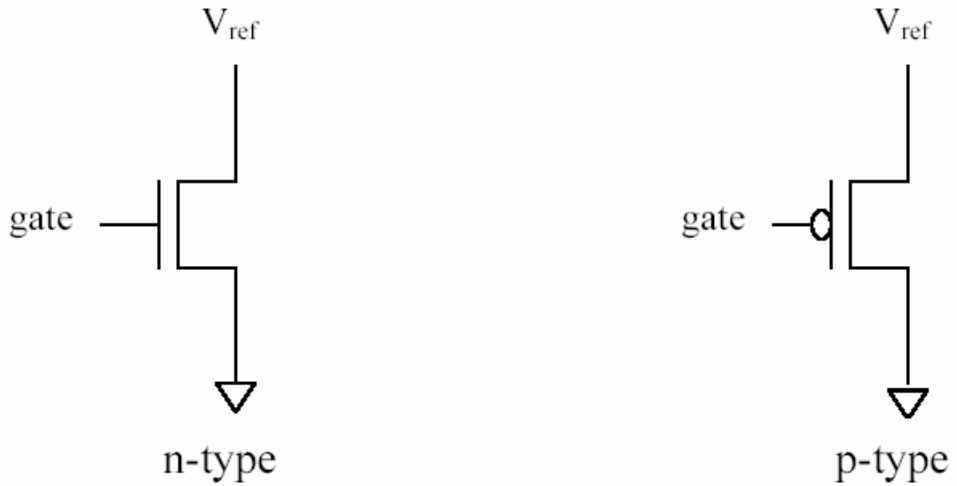
If the outputs of the previous NAND and NOR gates are fed to an inverter, we obtain the AND and OR gates. The five gates, (NOT, NAND, NOR, AND, and OR) are the principal building blocks of the digital logic level.

Because of the fewer transistors involved, many computers tend to be based on NAND and NOR gates rather than the more familiar AND and OR gates. However, the techniques required to describe and understand the circuits are the same no matter which gates are employed.

NAND and NOR gates are said to be "complete" because any Boolean function can be computed using NAND gates or NOR gates alone.

Digital Logic Structures

Most microprocessors today are constructed out of MOS transistors. MOS stands for *metal-oxide-semiconductor*. There are two types of MOS transistors: p-type and n-type. Their operation is analogous to that of ordinary wall switches. Symbolic representations of both types are shown below:



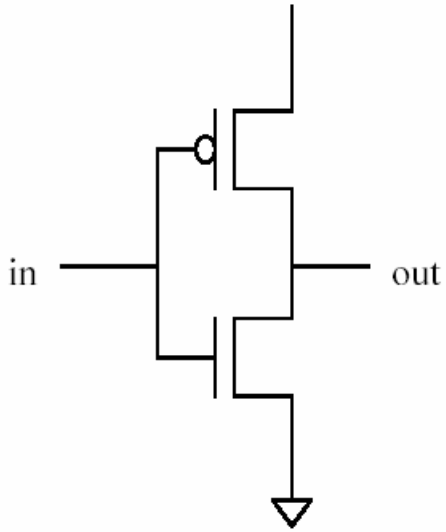
When the gate input of the n-type transistor is supplied with a sufficiently high voltage (e.g. 3 volts), the switch closes and the transistor essentially acts like a piece of wire. If the input voltage is less than the required value, then the switch opens and the connection between the reference voltage and the ground is broken.

Digital Logic Structures

The p-type transistor works in exactly the opposite fashion from the n-type transistor. When the gate input of the p-type transistor is 0 volts, the p-type transistor closes and creates a connection between the reference and the ground. When its input is sufficiently high, it opens and breaks the connection.

Because the n-type and p-type transistors act in this complementary way, circuits that contain both p-type and n-type transistors are referred to as CMOS circuits, *complementary metal-oxide-semiconductor.*

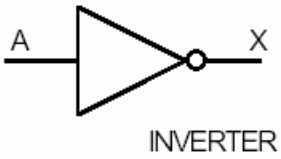
The diagram below shows how an inverter can be constructed using such transistors:



The Digital Logic Level

The simplest logic gates are shown below, along with a "truth table" for each that indicates the output generated by each gate as a function of the digital inputs:

NOT

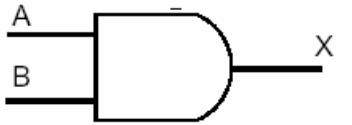


INVERTER

A	X
0	1
1	0

$X = \overline{A}$

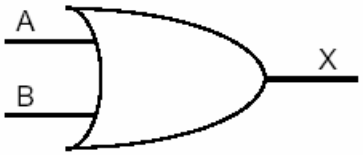
AND



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

X = A B

OR



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

X = A + B

XOR

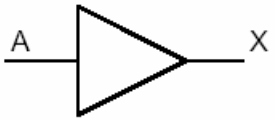
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

$X = A \oplus B$

Digital Logic Level

The logic gates shown below, can be produced by combining an inverter with the previously described gates. In many cases they may be simpler to implement than the more basic gates.

Buffer



A	X
0	0
1	1

$X = A$

NOR

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

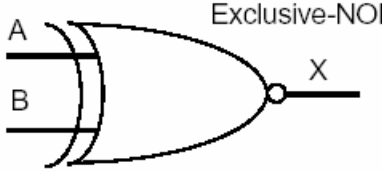
$X = \overline{A + B}$

NAND

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

$X = \overline{A\,B}$

Exclusive-NOR



A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

$X = \overline{(A \oplus B)}$

Integrated Circuits

Gates are usually manufactured in units called integrated circuits, which are also known as ICs or chips. These are square pieces of silicon about 5x5 mm on which some gates have been deposited.

ICs are typically mounted in rectangular plastic or ceramic packages measuring 5 to 15 mm wide by 20 to 50 mm long. The long sides of these rectangular packages contain rows of pins (about 5 mm in length) that can be inserted into sockets or soldered to printed circuit boards. These pins provide connections to the gates within the package, to a power source and to ground, etc.

These rectangular packages with the two rows of pins along the long edges are known as dual inline packages or DIPs. The number of pins varies as a function of the complexity of the chips. Larger chips are often placed into square packages with one or more rows of pins on all four sides.

Chips can be roughly classified based on the number of gates that they contain as follows:

SSI (Small Scale Integrated) circuits containing 1 to 10 gates.

MSI (Medium Scale Integrated) circuits containing 10 to 100 gates

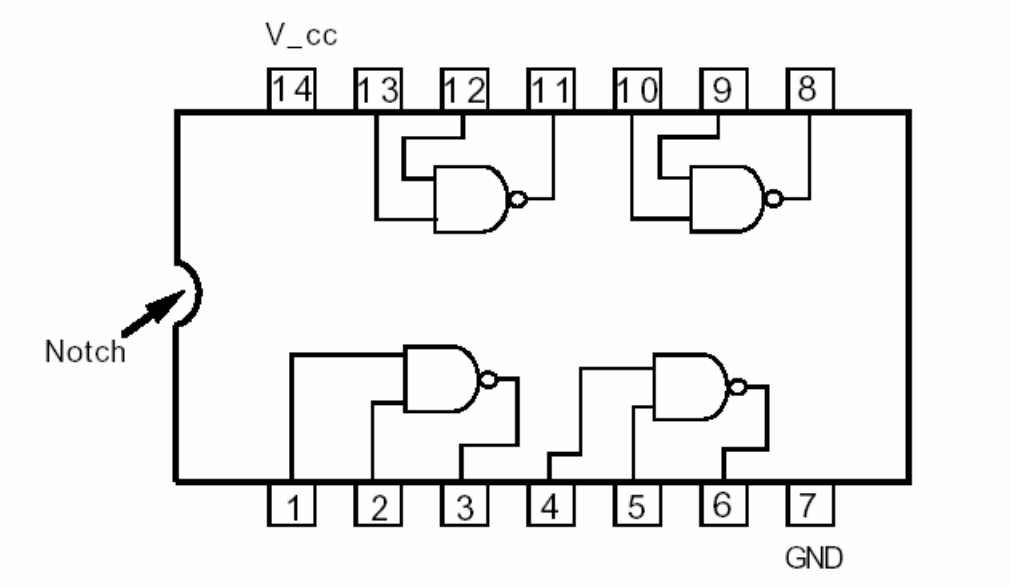
LSI (Large Scale Integrated) circuits containing 100 to 100,000 gates

VLSI (Very Large Scale Integrated circuits containing over 100,000 gates.

Current VLSI chips contain millions of transistors.

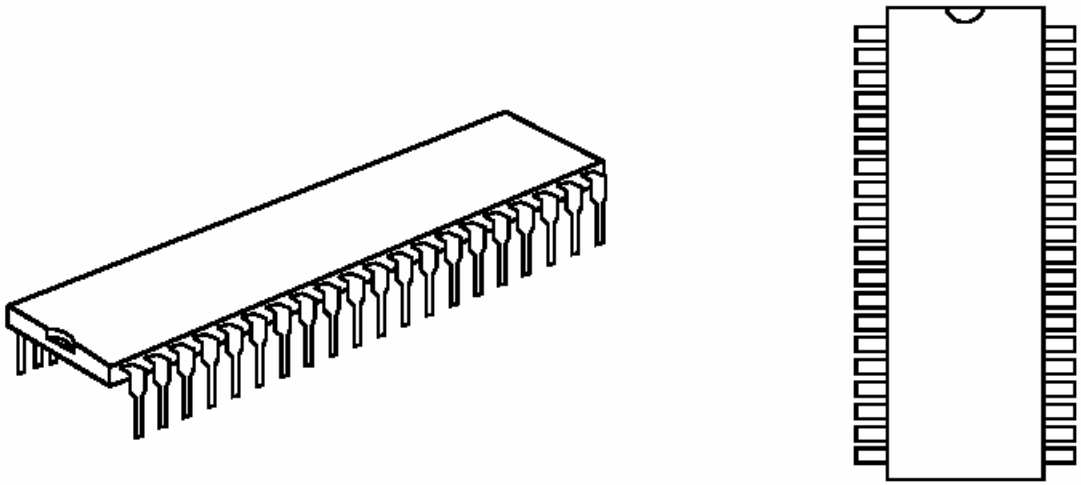
Integrated Circuits

The diagram below is a schematic drawing of a typical SSI chip containing four NAND gates. Each of the gates has two inputs and one output, requiring a total of 12 pins. In addition a pin is required for power and one for ground which are shared by all of the gates.



The packages generally have a notch near pin 1 to identify the orientation.

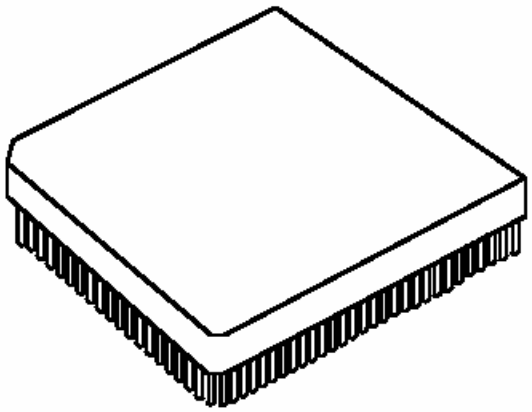
Sample IC Packaging Technologies

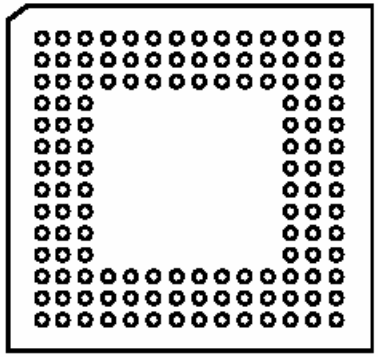


Dual in-line package (DIP)

Plastic quad flat pack (PQFP)

Sample IC Packaging Technologies





Pin grid array (PGA)

Boolean Algebra

Circuits built by combining logic gates may be described in terms of an algebra called Boolean algebra in which variables and functions can take on only the values 0 and 1.

Boolean functions take as input one or more Boolean variables and generate a result of 0 or 1. AND, OR, NOT and XOR are all examples of Boolean functions.

More complex Boolean functions may be defined as combinations of the basic functions such as AND , OR, etc. In general, a Boolean function of n variables has only 2^n possible sets of input values. For each combination of input values, the function generates either 0 or 1 as the result.

A Boolean function of n variables may be described by a truth table containing 2^n rows (one for each combination of input values). For example the addition of two Boolean variables is defined by

X	Y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

S is the sum of X plus Y

C is the carry generated by adding X plus Y

$S = \bar{X}Y + X\bar{Y}$ this is just the exclusive-or of X with Y

$S = X \oplus Y$ X XOR Y

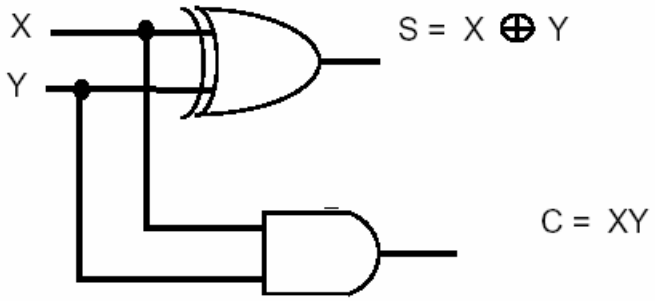
$C = XY$ that is X AND Y

The sum is 1 if exactly one of the inputs is 1, otherwise the sum is 0.

The carry is 0 unless both of the inputs are 1.

Half adders

A circuit that implements the addition function defined by the previous truth table is called a half adder. It can be constructed using the basic logic gates as follows:



The half adder works only for the low order bit within the sum. The higher order bits in the sum would require that the carry from the bit position immediately to the right be taken into account.

If CI is the carry in and CO is the carry out of a particular bit position, then the truth table for a "full adder" is:

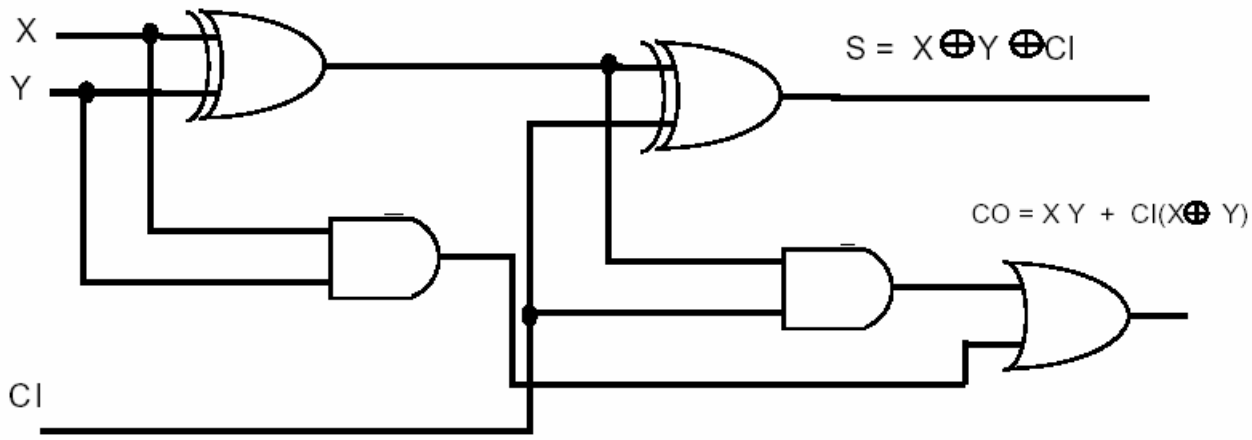
X	Y	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S = \bar{X}\,\bar{Y}\,CI + \bar{X}\,Y\,\bar{CI} + X\,\bar{Y}\,\bar{CI} + X\,Y\,CI = X \oplus Y \oplus CI$$

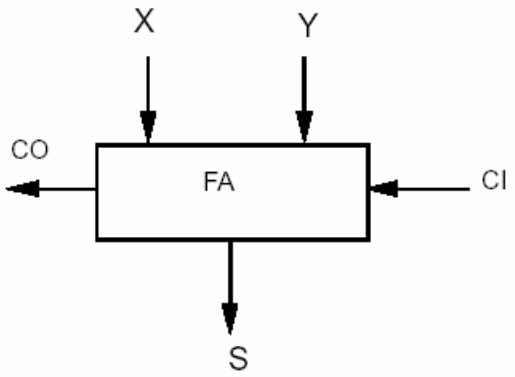
$$CO = \bar{X}\,Y\,CI + X\,\bar{Y}\,CI + X\,Y\,\bar{CI} + X\,Y\,CI = CI(X \oplus Y) + XY$$

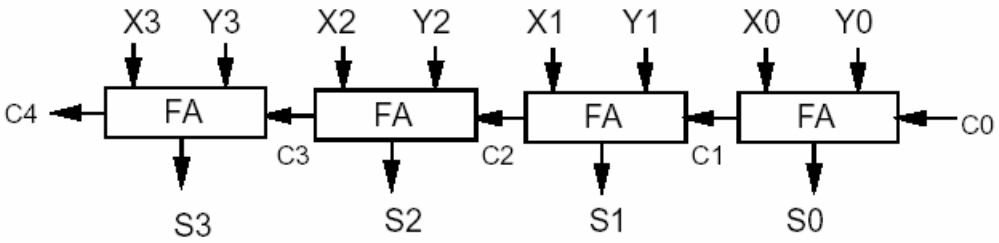
Full adders

A full adder may be constructed by combining two half adders as follows:



For our purposes the full adder may be thought of as a black box that takes as input X, Y, and CI and generates S and CO as output.



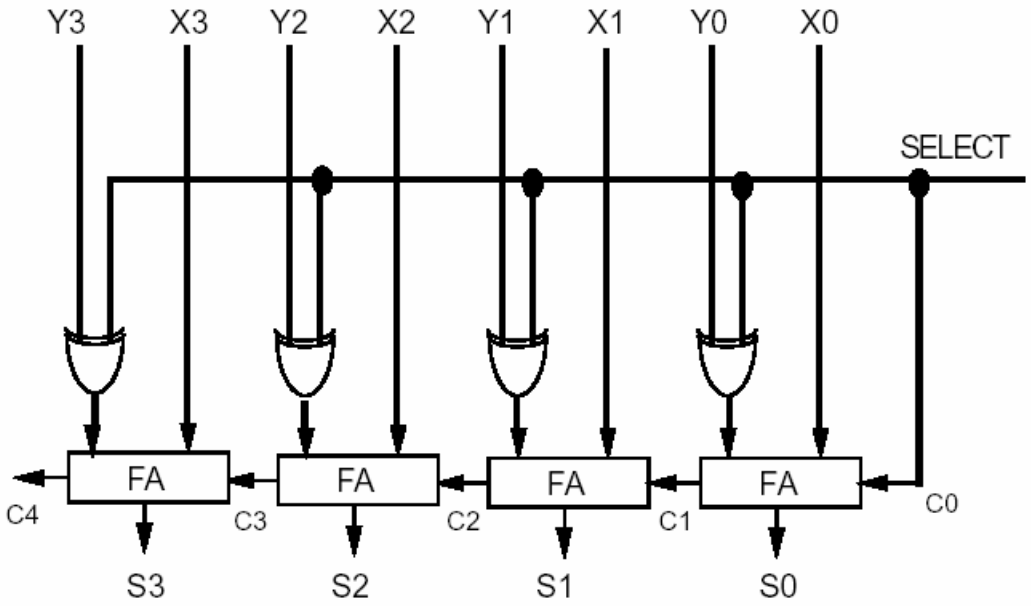


(4-BIT Parrallel adder with ripple carry)

Binary Adder-Subtractor

The subtraction of binary numbers can be performed most conveniently by adding the two's complement of the subtrahend to the menuend to yield the two's complement difference between the two numbers.

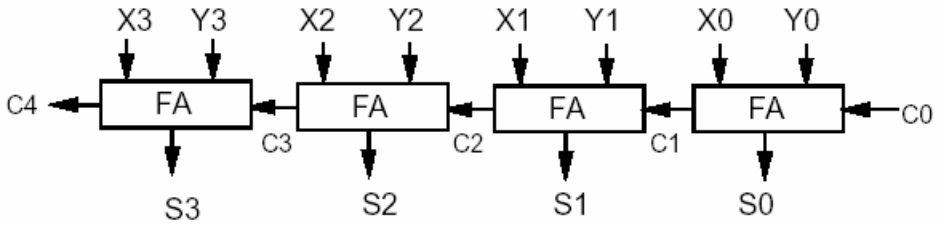
A circuit for subtracting X-Y may be constructed from a parrallel adder with inverters placed between each Y terminal and the corresponding full adder.



The addition and subtraction operations can be combined into one circuit with one common binary adder. Shown above is a 4-bit adder-subtractor. When SELECT=0 the circuit is an adder and when SELECT = 1, the circuit becomes a subtractor.

Each exclusive-OR gate receives SELECT and one of the bits from Y. When SELECT=0 the XOR gate simply outputs the original bit from Y (since 0 XOR 0 = 0 and 0 XOR 1 = 1). Note that C0 = SELECT, so when SELECT = 0, the sum of X and Y is generated. But when SELECT = 1, each bit within Y is inverted (yielding the one's complement of Y) which is then added to X along with C0 (which now equals 1) , thus yielding the sum of X and the two's complement of Y. The result is therefore the same as that produced by subtracting Y from X.

The Look-Ahead-Carry Adder

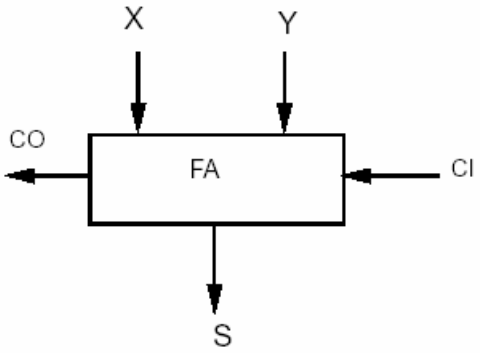


(4-BIT Parrallel adder with ripple carry)

The speed of the ripple carry adder is limited by the time required for the carries to propagate through all stages of the adder.

The look-ahead-carry adder speeds up this addition process by eliminating this ripple carry delay.

Consider the operation of the full adder.



The Look-Ahead-Carry Adder

There are two ways in which an output carry is produced by the full adder. If both X and Y are 1s, then the adder is said to **generate** an output carry (CO=1).

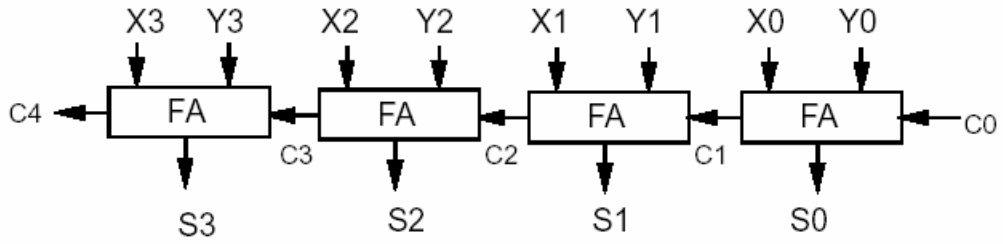
Thus $C_g = XY$ i.e. X AND Y

If either X or Y is 1, then an input carry CI will be **propagated** , so

$$C_p = X + Y$$

Hence $CO = C_g + (C_p\ CI)$

The Look-Ahead-Carry Adder



Applying this relation to each of the full adders yields

$C1 = Cg1 + Cp1\ C0$

$C2 = Cg2 + Cp2\ C1 = Cg2 + Cp2\ (Cg1 + Cp1\ C0) = Cg2 + Cp2\ Cg1 + Cp2\ Cp1\ C0$

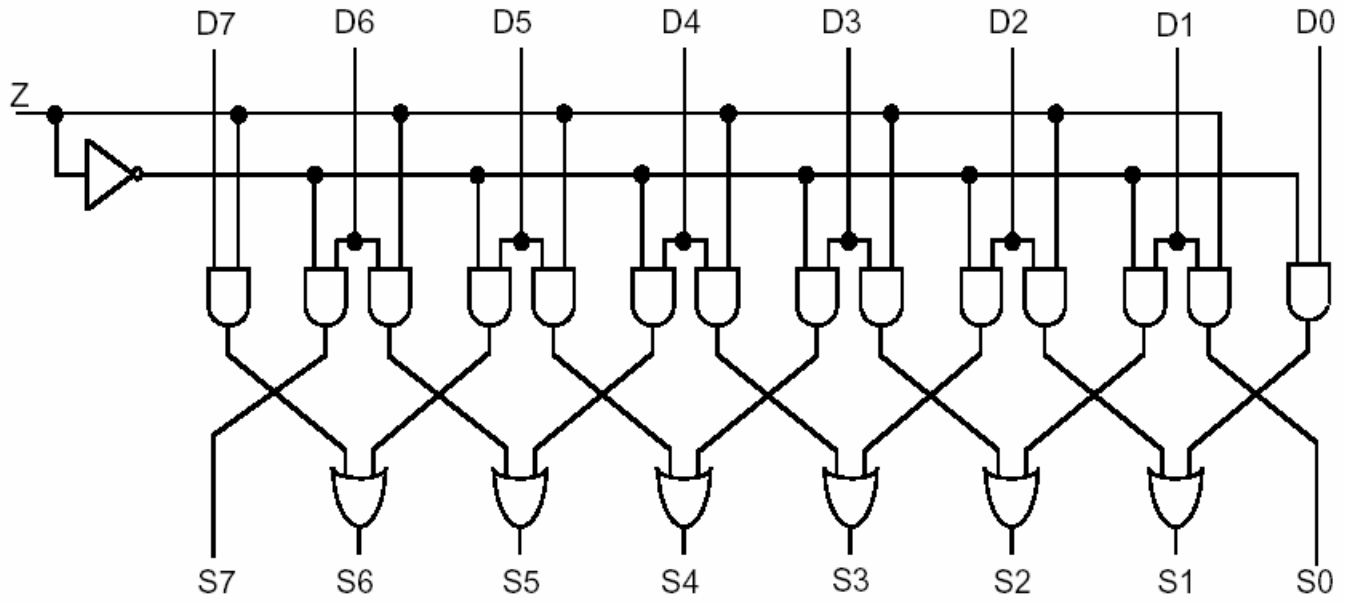
$C3 = Cg3 + Cp3\ C2 = Cg3 + Cp3\ (Cg2 + Cp2\ Cg1 + Cp2\ Cp1\ C0)$
$= Cg3 + Cp3\ Cg2 + Cp3\ Cp2\ Cg1 + Cp3\ Cp2\ Cp1\ C0$

$C4 = Cg4 + Cp4\ C3 = Cg4 + Cp4\ (Cg3 + Cp3\ Cg2 + Cp3\ Cp2\ Cg1 + Cp3\ Cp2\ Cp1\ C0)$
$= Cg4 + Cp4\ Cg3 + Cp4\ Cp3\ Cg2 + Cp4\ Cp3\ Cp2\ Cg1 + Cp4\ Cp3\ Cp2\ Cp1\ C0$

Hence the output carry of each stage is dependent only on the initial input carry (C0), the Cg and Cp functions of that stage, and the Cg and Cp functions of the preceding stages. Since each of the Cg and Cp functions can be expressed in terms of the X and Y inputs, all of the output carries are immediately available so there is no need to wait for the carry to ripple through all stages before the final result is produced. Thus the bits in the sum are essentially produced in parallel.

A binary shifter

As another example of the use of logic gates to carry out operations within the CPU, consider the following circuit:



The control line, Z, determines the direction of the shift, 0 for left and 1 for right. The eight inputs are presented on lines D0,..., D7. The output, which is just the input shifted one bit, is available on the S0,..., S7 lines.

When Z =1, the right member in each pair of AND gates passes the input bit Di to its output. Since the output of the right AND gate is wired to the input of the OR gate on its right, a right shift is performed. When Z=0, the left AND gate passes its input on to the OR gate on its left causing a left shift.

Implementation of Boolean Functions

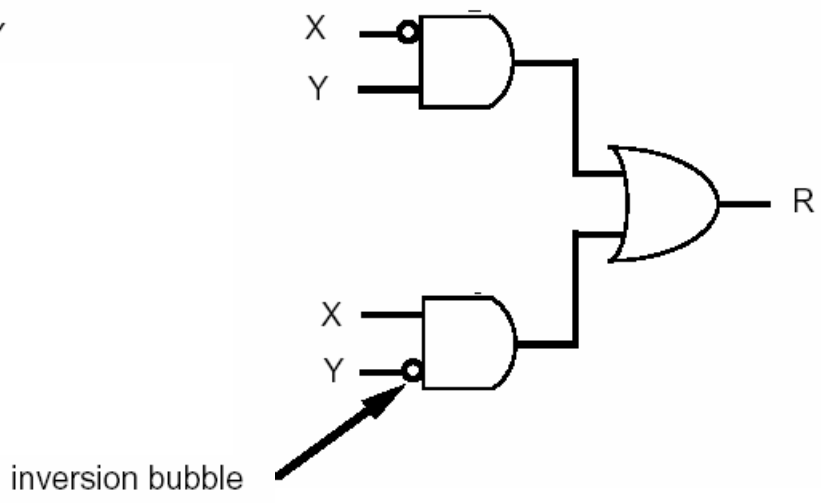
The following procedure can be used to implement a Boolean function from the basic logic gates:

1. Write down the truth table for the function.
2. Provide inverters to generate the complement of each input.
3. Draw an AND gate for each term with a 1 in the result column.
4. Wire the AND gates to the appropriate inputs.
5. Feed the output of all AND gates into an OR gate to generate the result.

Example: Recall that the exclusive OR function is defined as:

$$R = \overline{X}Y + \overline{Y}X = X \oplus Y$$

X	Y	R
0	0	0
0	1	1
1	0	1
1	1	0



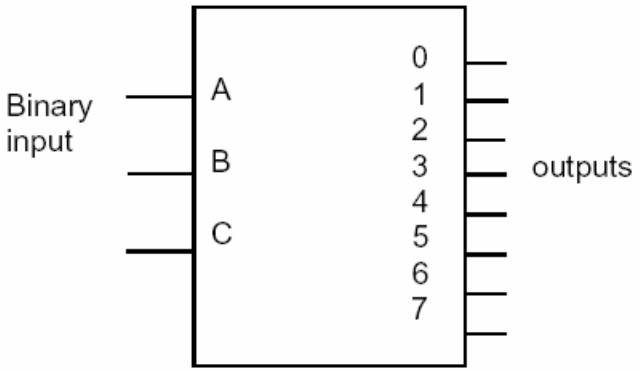
Boolean Algebra

Some identities of Boolean algebra

Name	AND Form	OR Form
Identity law	$1A = A$	$0 + A = A$
Null law	$0A = 0$	$1 + A = 1$
Idempotent law	$A\,A = A$	$A + A = A$
Inverse law	$A\,\overline{A} = 0$	$A + \overline{A} = 1$
Commutative law	$A\,B = B\,A$	$A + B = B + A$
Associative law	$(AB)C = A(BC)$	$(A+B) + C = A + (B + C)$
Distributive law	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
Absorption law	$A(A + B) = A$	$A + AB = A$
De Morgan's law	$\overline{AB} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$

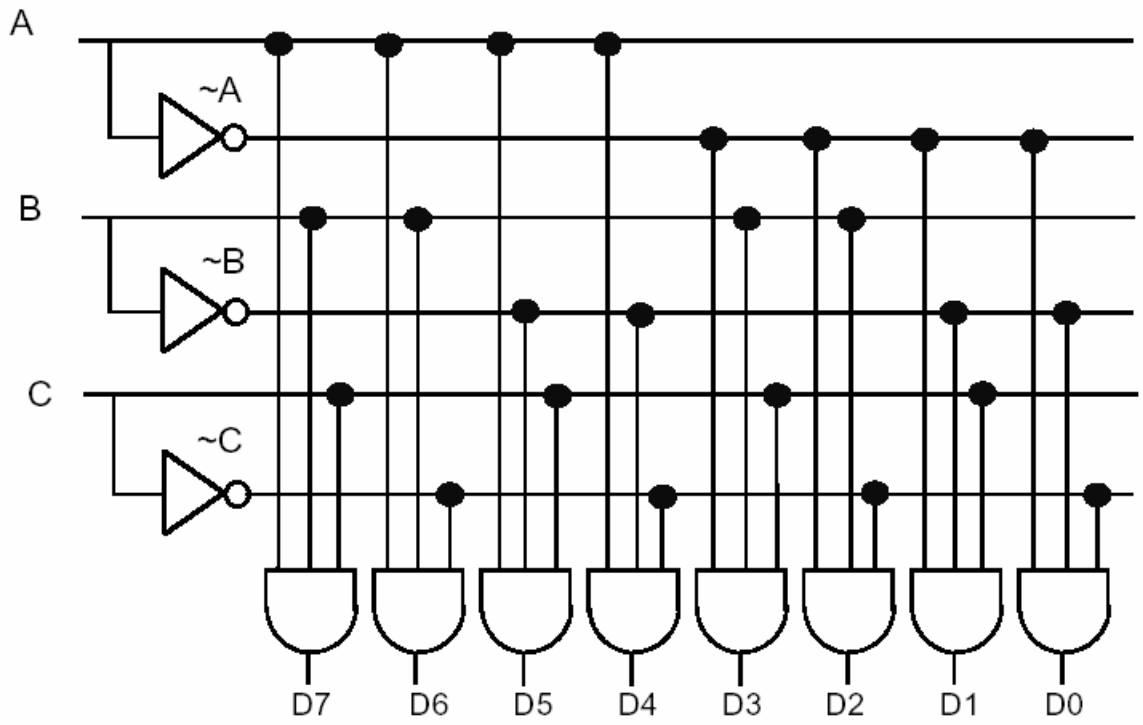
Decoders and Multiplexors

Circuits that transform binary encoded values (such as register numbers or ALU operations) into a single control signal as output are referred to as decoders. For example, a decoder that takes a 3-bit input number would generate a signal on one of 8 output lines as shown below:



INPUTS	OUTPUTS
A B C	0 1 2 3 4 5 6 7
0 0 0	1 0 0 0 0 0 0 0
0 0 1	0 1 0 0 0 0 0 0
0 1 0	0 0 1 0 0 0 0 0
0 1 1	0 0 0 1 0 0 0 0
1 0 0	0 0 0 0 1 0 0 0
1 0 1	0 0 0 0 0 1 0 0
1 1 0	0 0 0 0 0 0 1 0
1 1 1	0 0 0 0 0 0 0 1

A possible implementation of this decoder using logic gates is shown below:

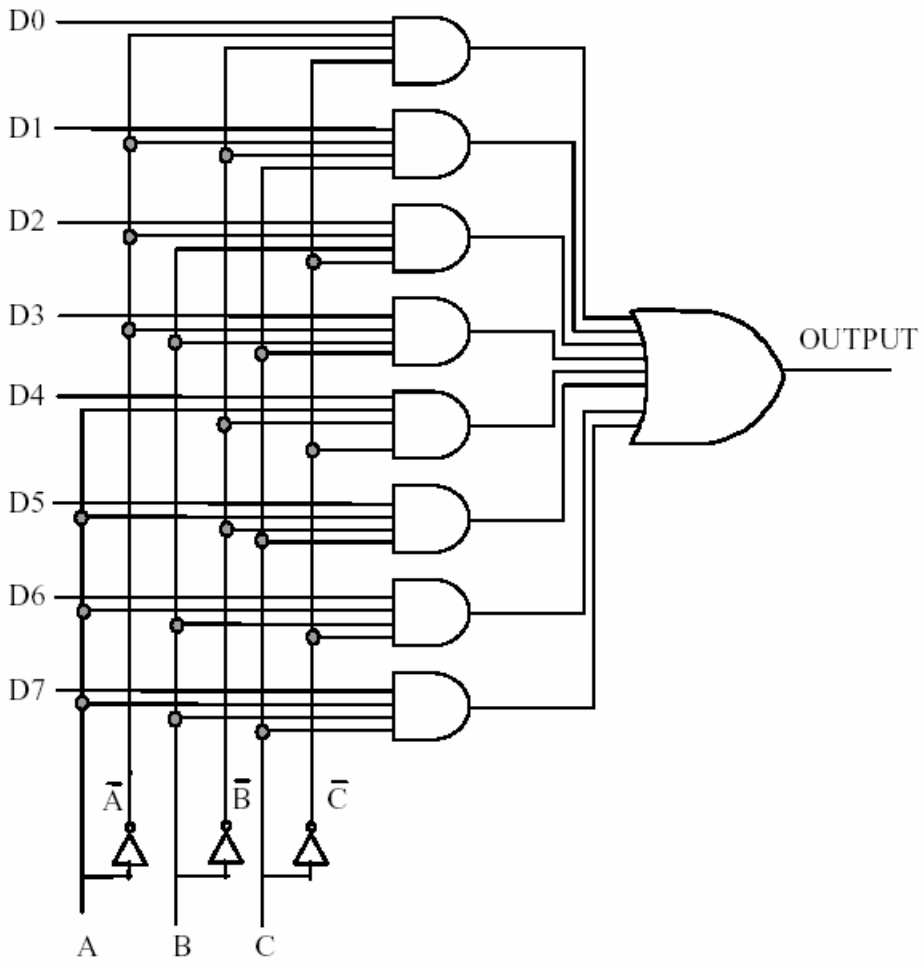


Decoders and Multiplexers

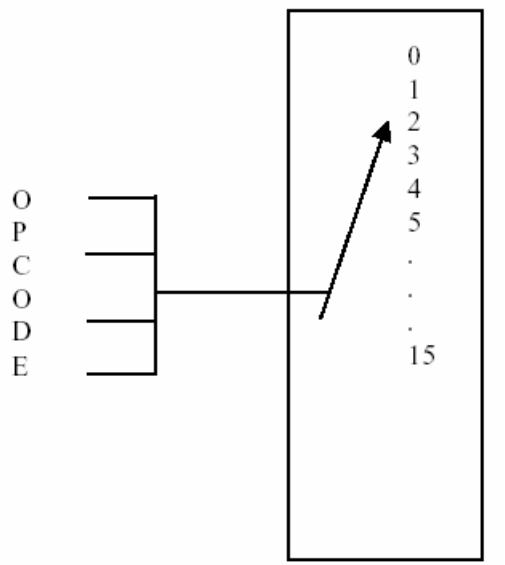
Similarly a 4-bit decoder could be used to generate select signals for one of 16 registers or to select one of up to 16 ALU operations.

The inverse of a decoder is a multiplexer. This is a digital circuit that takes as input 2^n digital signals and n control inputs that select one of the input signals to be "gated" (i.e. routed) to the single output.

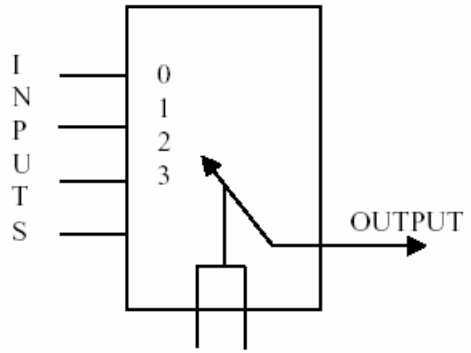
Shown below is a 8-input multiplexer. The three control lines A, B, and C, encode a 3-bit number that specifies which of the eight input lines is to be gated to the OR gate and therefore to the output. No matter what value is on the control lines, seven of the AND gates will always output 0; the other one may output either 0 or 1, depending on the value of the selected input line. Each AND gate is enabled by a different combination of the control inputs.



Decoders and Multiplexers



ALU Operation Decoder



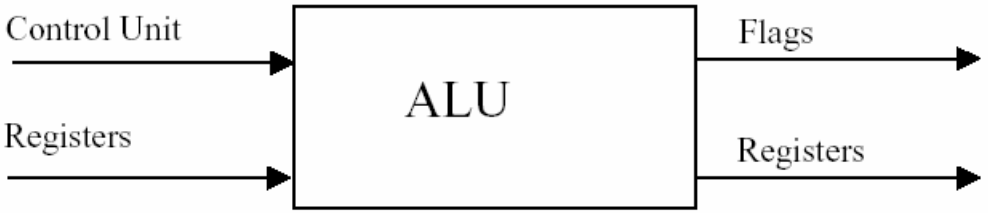
Multiplexer Operation

The Arithmetic Logic Unit and Computer Arithmetic

The ALU is that part of the computer that performs operations such as addition, subtraction, multiplication and division on data. Besides these arithmetic operations, the ALU also performs shifts, AND's, OR's and carries out other logic operations on data as well.

The other components of the computer system such as the control unit, registers, memory, and the I/O system are there mainly to bring data into the ALU for it to process and then to take the results back out.

The functions performed by the ALU are implemented via combinations of simple logic devices. The ALU could be viewed as a black box to which data are provided via registers and whose operation is supervised by signals from the control unit. The results of an operation are stored into registers by the ALU and flags, such as the condition bits within some status register, are set by the ALU to indicate the current status.



The operation to be performed must be decoded from signals transferred to the ALU form the control unit. Thus the ALU must contain a number of logic circuits to accomplish its functions. These include:

Adders/subtractors

Shifters

Multipliers

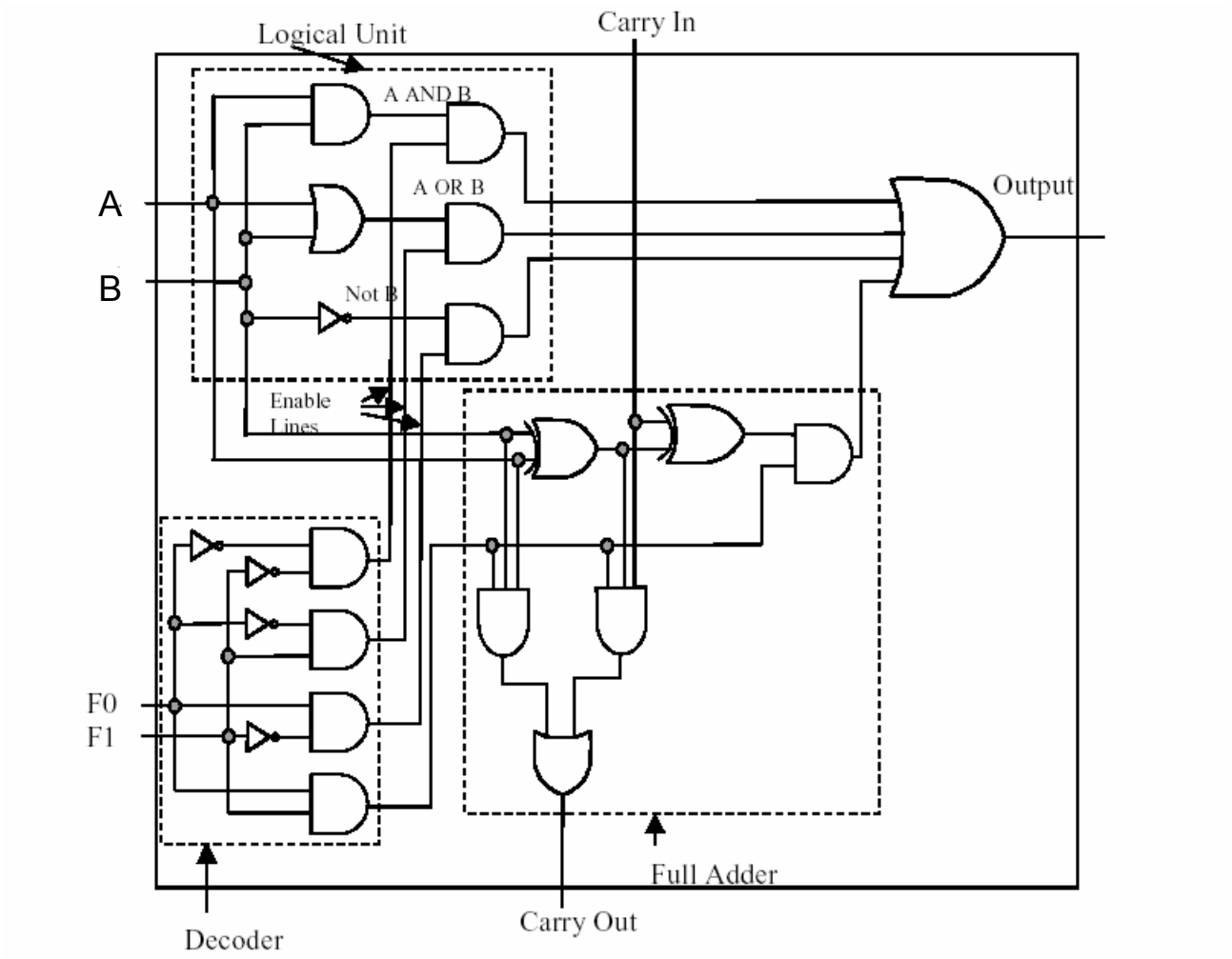
Decoders

Multiplexers

Comparators

ALU Implementation

The circuit shown below is a simple one-bit ALU. It is capable of computing one of four functions – namely, A AND B, A OR B, Not B, and the sum of A + B. The particular function connected to the output depends on the selected signals F0 and F1 (00 for AND, 01 for OR, 10 for Not, and 11 for the sum).



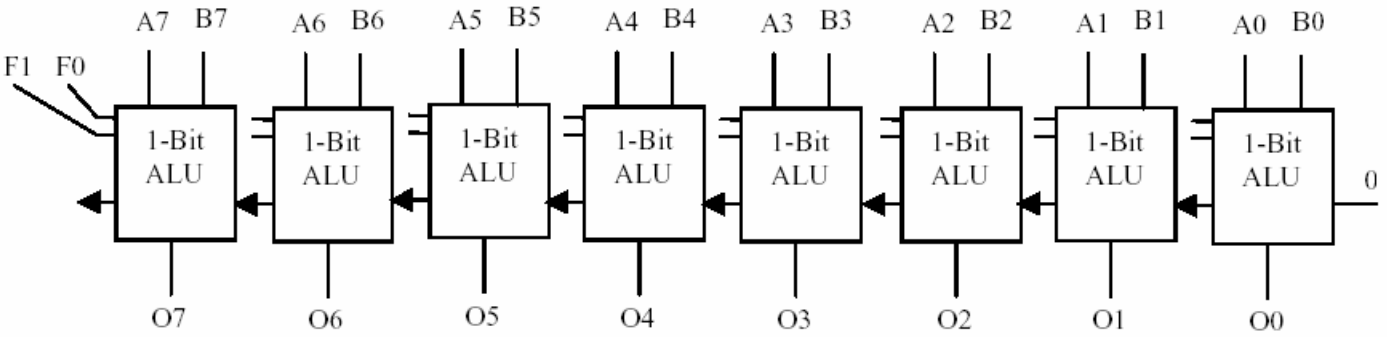
ALU Implementation

The lower left corner of the ALU contains a 2-bit decoder to generate enable lines for the four operations based on F0 and F1. The upper left corner has the logic to compute A AND B, A OR B, and Not B, but at most one of these is passed onto the final OR gate via the enable lines coming out of the decoder.

The lower right corner of the ALU contains a full adder for computing the sum of A and B. Chips containing circuits such as this are called **bit slices**. Such circuits can be used to build a ALU of any desired width.

A computer designer could use chips containing several such circuits to build an ALU of any desired width.

Example: an 8-bit ALU could be produced by connecting eight 1-bit ALU slices.



However, it is more common for a chip to contain multiple slices (e.g. 4 or 8) packaged together. This tends to simplify the design and to reduce the chip count. For example, four chips each containing eight slices could be connected to build a 32-bit ALU.

Combinational and Sequential Logic

The circuits that have been examined up to now have all been examples of combinational logic circuits. These are defined as circuits whose output at some time T depend only on the inputs at time T. That is, there is no dependence on the previous history of inputs to the circuit.

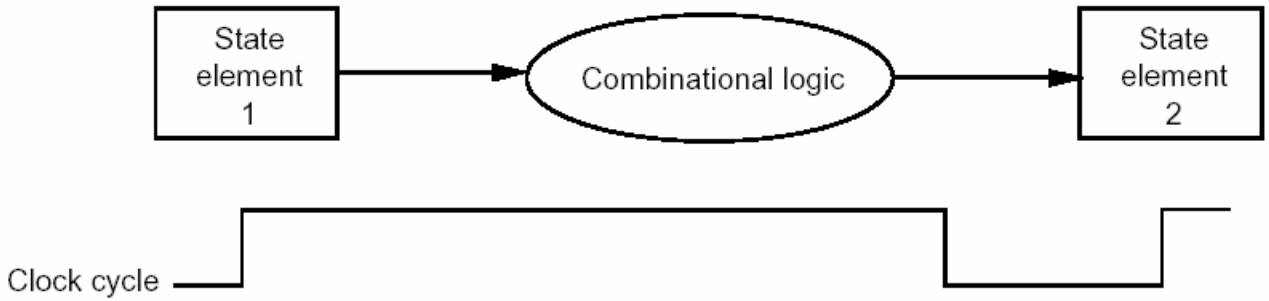
Another type of circuit is exemplified by flip-flops, registers and memories. The output of these circuits depend not only on the inputs at the current time, but also on the previous inputs. Such circuits are referred to as sequential logic circuits. And the devices comprised of such circuits are sometimes called state elements. These types of circuits are said to have "memory" because they behave as though they remember the previous inputs.

The current state of a system is determined by the content of these state elements. If after having been powered down, the state elements are restored to there previous condition, a computer system would operate as though it had never lost power.

Combinational and Sequential Logic

The outputs generated by combinational circuits may be saved by inputting these values to some type of sequential circuit (state element).

In a synchronous system a clock is used to control when a state element samples its input. The inputs to the state element must have sufficient time to reach a stable value before they are sampled. This stablization time affects the minimum clock period that can be used.

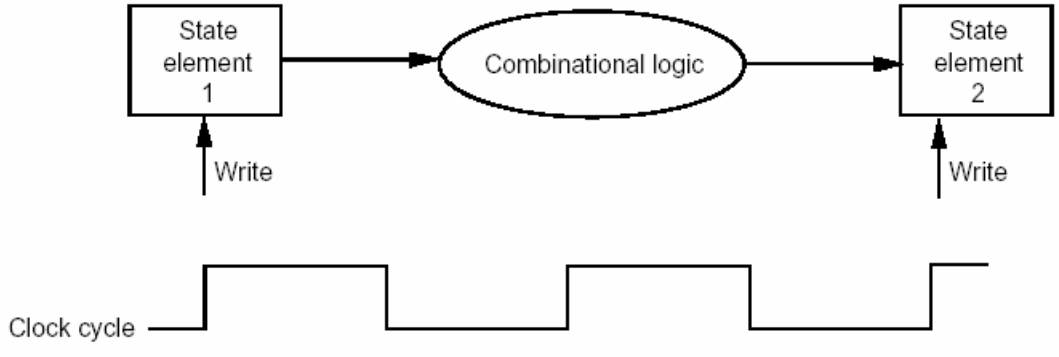


Combinational and Sequential Logic

The main disadvantage with the synchronous system is that the basic clock period has to be long enough to accommodate the slowest circuit in the system.

An alternative is to allow the various circuits to use a varying number of clock cycles. The faster circuits would need fewer cycles while the more time consuming functions would take multiple clock cycles.

However, with this technique a separate mechanism would be required to identify the number of clock cycles needed. That is, on which clock edge the inputs to a state element should be sampled. This situation is illustrated below:



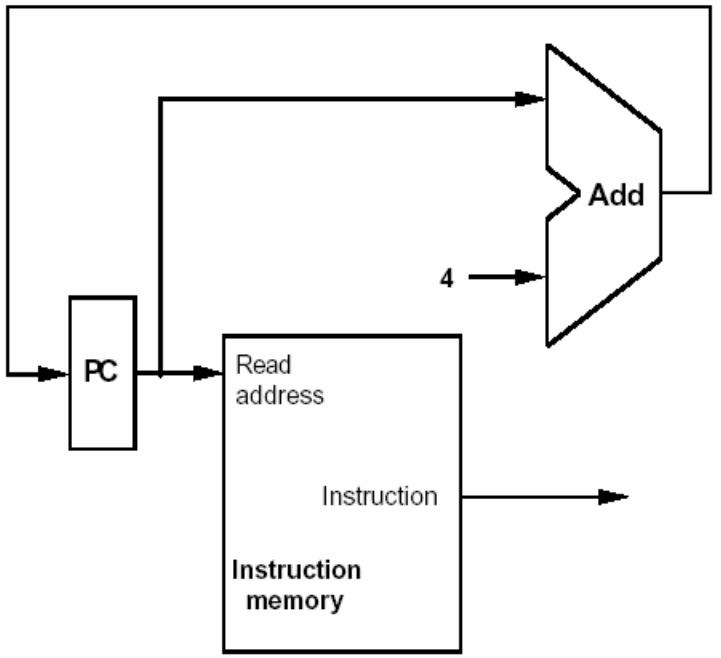
In this case, the combinational circuit can take several cycles to use the inputs from state element 1 in generating the inputs for state element 2. The write control signal for state element 1 must be "deasserted" to prevent the inputs to the combinational circuit from changing until its result has been sampled and stored by state element 2.

Signals such as the write signals for these state elements may be generated by the control unit within the computer.

The Datapath

The data path consists of the components that supply instructions and data and that perform the operations called for by each instruction on the data operands. These include

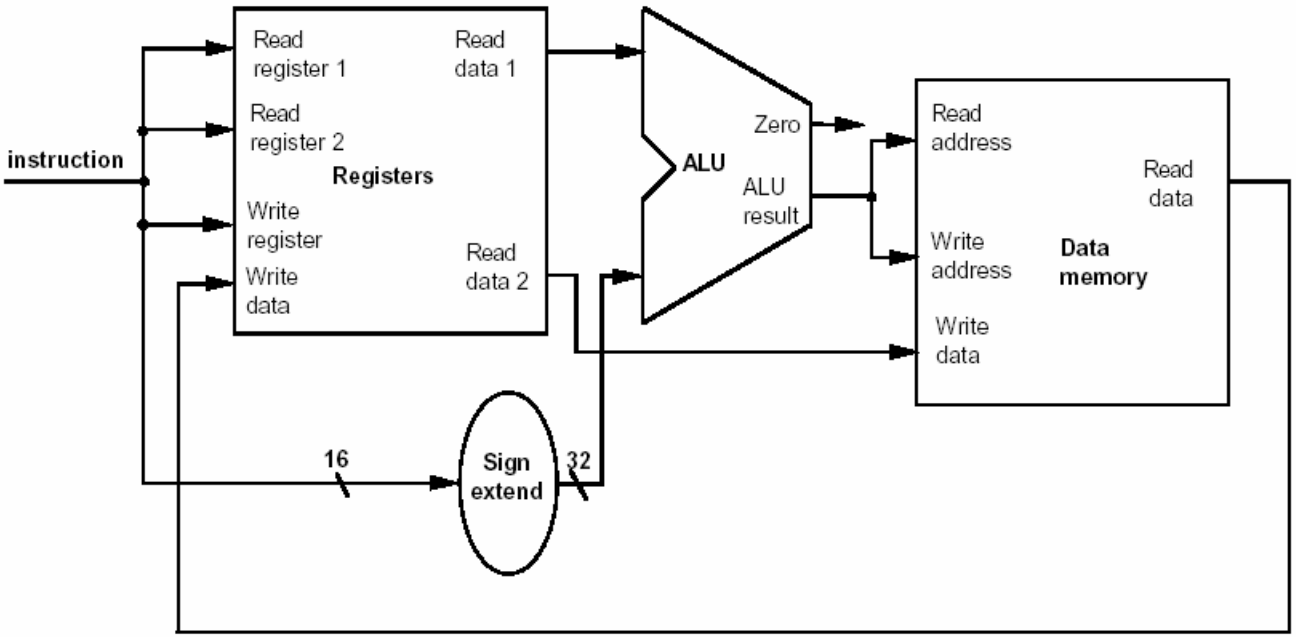
the memory
the register file
the program counter (PC)
the instruction register (IR)
the ALU, adders, shifters, sign-extension unit, etc.
the buses that interconnect the components within the data path



The control unit is responsible for generating the required signals to activate the data path components at the appropriate time. The components used depend on the type of instruction that is being executed. A separate adder can be used to increment the PC to allow sequential access of instructions.

The Datapath

The diagram below shows a data path that can be used to execute either R-type instructions or memory access instructions.



For memory access instructions the sign-extended displacement is added to a register to generate the address of the memory location to be referenced.

For R-type instructions, two registers are used as the operands for the operation specified by the op-code and function fields within the instruction.

A multiplexor is used to select between a register and the sign-extended displacement as the second operand to the ALU. Another multiplexor is used to select between the output of the ALU and the output of the data memory to be written back to the register file. For lw instructions the output of the data memory is selected, while the output of the ALU is selected for R-type instructions.

The Instruction Execution Cycle

A number of steps are involved in the execution of any instruction. As a basis for describing the instruction execution cycle, as well as the operation of the datapath and the control unit, a subset of the MIPS instruction repertoire will be used. This subset is made up of:

- the memory reference instructions *load word (lw)* and *store word (sw)*
- the arithmetic-logic instructions *add, sub, and, or,* & *slt*
- the branch on equal *(beq)* and the jump instruction *(j)*

The initial steps in executing all of these instructions are the same:

1. the program counter (PC) is used to fetch the instruction from memory
2. the PC is incremented by 4 to point to the next instruction in sequence
3. one or two registers are read (indicated by fields within the instruction).
   Load instructions read 1 register while the others read two registers.

Step 4 requires the use of the ALU, but the specific function required of the ALU depends on the type of instruction. The ALU is used to

- compute the effective address for memory reference instructions
- perform the operation for arithmetic-logic instructions
- perform a comparison for branch instructions

The Instruction Execution Cycle

For step 5,

- a memory reference instruction must access memory
- an arithmetic-logic instruction must store the result into a register
- a branch, if taken, must replace the contents of the PC with the branch address

The branch target address is computed by sign extending the 16-bit displacement field from the instruction and combining this number with the incremented PC.

The branch instruction must also perform a compare by sending two arguments to the ALU and requesting a subtraction operation. The outcome of this subtraction is indicated by the Zero signal produced by the ALU.

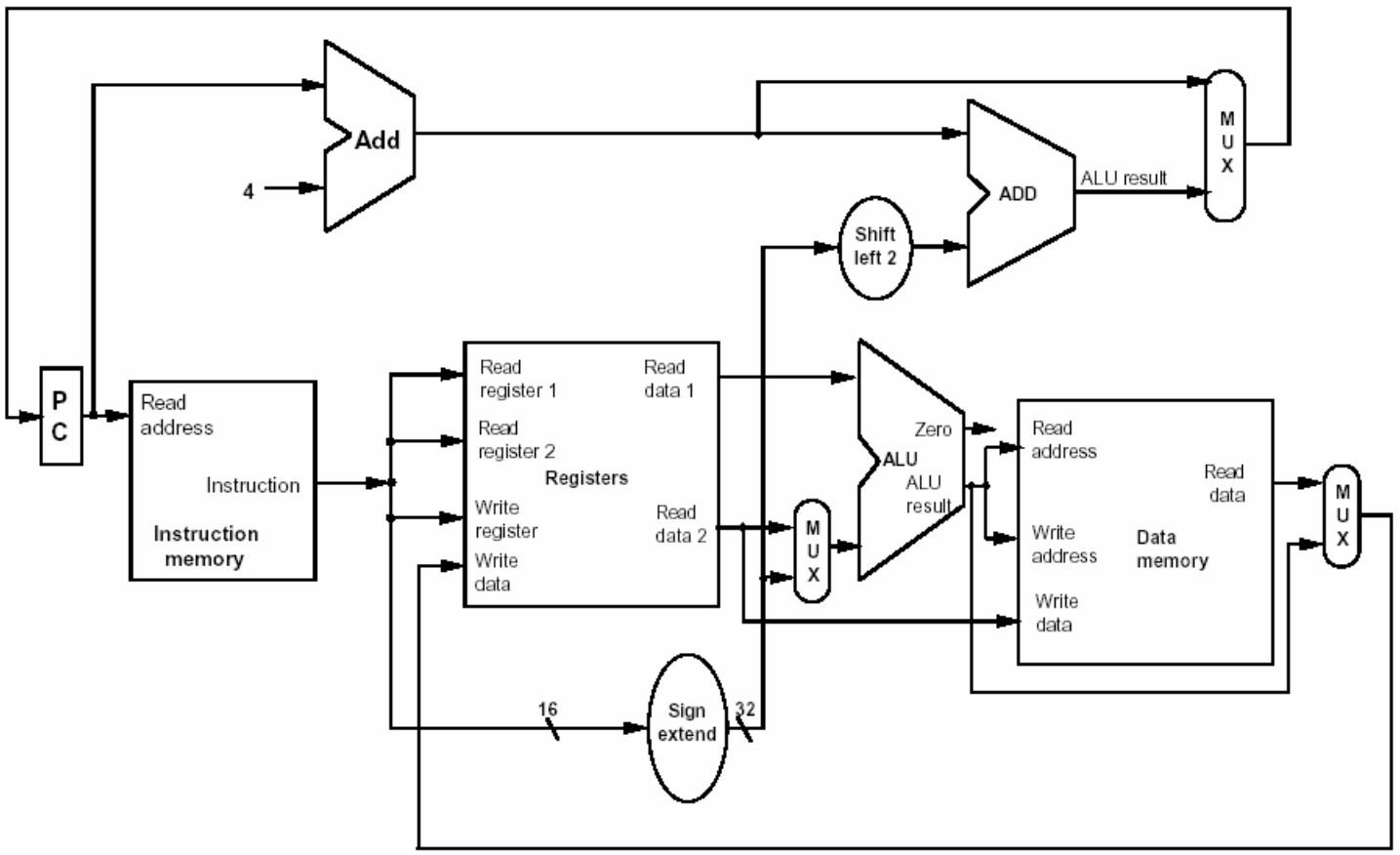
If the result is zero, then the computed branch address replaces the value in the PC, otherwise the new contents of the program counter remains PC+4, i.e., the address of the instruction immediately following the branch instruction.

MIPS Instruction Subset Implementation

For each instruction, the ALU will have to perform one of five possible functions. The function performed, is determine by the 3-bit ALU control input:

ALU control input	Function
000	And
001	OR
010	Add
110	Subtract
111	Set-on-less-than

The 3-bit ALU control input is generated by the control unit in response to the contents of the op-code and function fields within the instruction.



Simple data path for fetching and executing load/store, arithmetic and branch instructions.

The Instruction Execution Cycle

To see what additional signals must be generated by the control unit, recall the format of the MIPS instructions:

R-type

0	rs	rt	rd	shamt	funct
31 - 26	25 - 21	20 - 16	15 - 11	10 - 6	5 - 0

rs and **rt** specify the two registers to be read as operands,
rd specifies the register into which the result is written

Load or store instruction format

35 or 43	rs	rt	address
31 - 26	25 - 21	20 - 16	15 - 0

rs specifies the base register to which the sign extended address field is added to generate the address of the memory operand.
rt specifies the register to receive the memory operand for loads or the register that contains the value to be written into memory for stores

Branch instruction format

4	rs	rt	address
31 - 26	25 - 21	20 - 16	15 - 0

rs and **rt** specify the two registers to be compared. If the registers are equal, then the sign extended address field is added to the PC.
The registers are compared by subtracting rs from rt and testing the result for zero. The Zero control signal generated by the ALU will be asserted if the result is zero indicating equality.

Jump instruction format

2	address
31 - 26	25 - 0

The address is shifted left 2 bits and concatenated to bits 28 through 31 of the incremented PC to yield the 32-bit jump address.