Abstraction Levels

Stephen A. Edwards

Computer Science Pedagogy
Drew University

July 30, 2025

(PRAXIS 5652: Computer Sciences Sample Test Questions)

4. Consider the following list.

- Assembly language
- Block-based programming language
- Logic gate
- Machine language

Which of the following arranges the list in order from highest level of abstraction to lowest level of abstraction?

A. Block-based programming language, assembly language, machine language, logic gate

B. Block-based programming language, machine language, assembly language, logic gate

C. Block-based programming language, machine language, logic gate, assembly language

D. Machine language, block-based programming language, assembly language, logic gate

What's the largest number that divides both 6 and 15?

(The Greatest Common Divisor)

Euclid's algorithm for finding the greatest common divisor:

1. Start with two numbers
2. If the numbers are the same, stop: there's your answer
3. Subtract the smaller number from the bigger number
4. Go to step 2

Euclid's algorithm for finding the greatest common divisor:

1. Start with two numbers
2. If the numbers are the same, stop: there's your answer
3. Subtract the smaller number from the bigger number
4. Go to step 2

In Python,

```
while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a
```

Euclid's algorithm for finding the greatest common divisor:

1. Start with two numbers
2. If the numbers are the same, stop: there's your answer
3. Subtract the smaller number from the bigger number
4. Go to step 2

In Python,

```
while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a
```

In decimal,

15 6 Two numbers

Euclid's algorithm for finding the greatest common divisor:

1. Start with two numbers
2. If the numbers are the same, stop: there's your answer
3. Subtract the smaller number from the bigger number
4. Go to step 2

In Python,

```
while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a
```

In decimal,

15	6	Two numbers
9	6	After subtracting 6 from 15
3	6	After subtracting 6 from 9
3	3	After subtracting 3 from 6
		We're done

Euclid's algorithm for finding the greatest common divisor:

1. Start with two numbers
2. If the numbers are the same, stop: there's your answer
3. Subtract the smaller number from the bigger number
4. Go to step 2

In Python,

```
while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a
```

In decimal,

15	6	Two numbers
9	6	After subtracting 6 from 15
3	6	After subtracting 6 from 9
3	3	After subtracting 3 from 6
		We're done

In four-bit binary,

1111 0110

Euclid's algorithm for finding the greatest common divisor:

1. Start with two numbers
2. If the numbers are the same, stop: there's your answer
3. Subtract the smaller number from the bigger number
4. Go to step 2

In Python,

```
while a != b:
    if a > b:
        a = a - b
    else:
        b = b - a
```

In decimal,

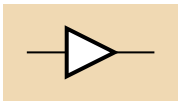
15	6	Two numbers
9	6	After subtracting 6 from 15
3	6	After subtracting 6 from 9
3	3	After subtracting 3 from 6
		We're done

In four-bit binary,

1111	0110
1001	0110
0011	0110
0011	0011

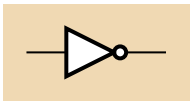
The Menagerie of Logic Gates

Buffer



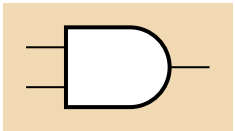
0	0
1	1

Inverter



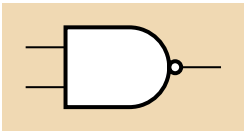
0	1
1	0

AND



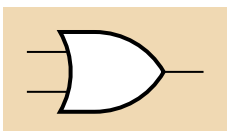
$\cdot$	0	1
0	0	0
1	0	1

NAND



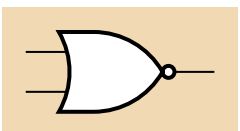
$\overline{\cdot}$	0	1
0	1	1
1	1	0

OR



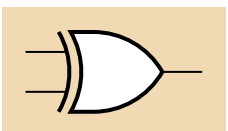
$+$	0	1
0	0	1
1	1	1

NOR



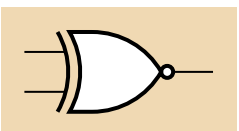
$\overline{+}$	0	1
0	1	0
1	0	0

XOR



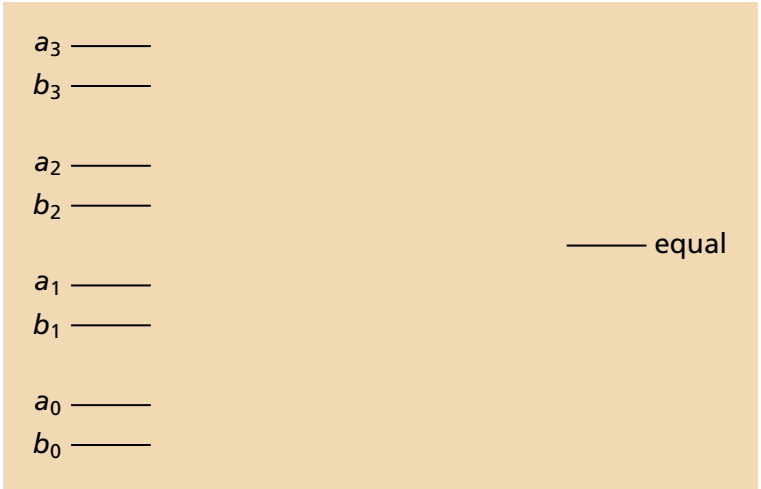
$\oplus$	0	1
0	0	1
1	1	0

XNOR

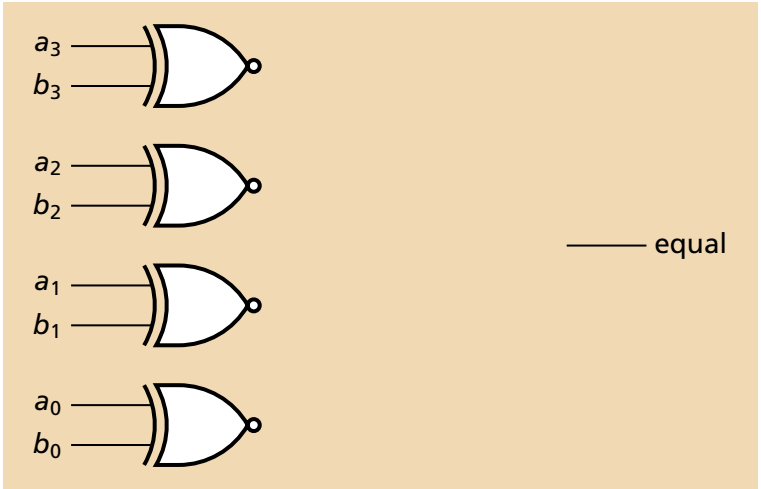


$\overline{\oplus}$	0	1
0	1	0
1	0	1

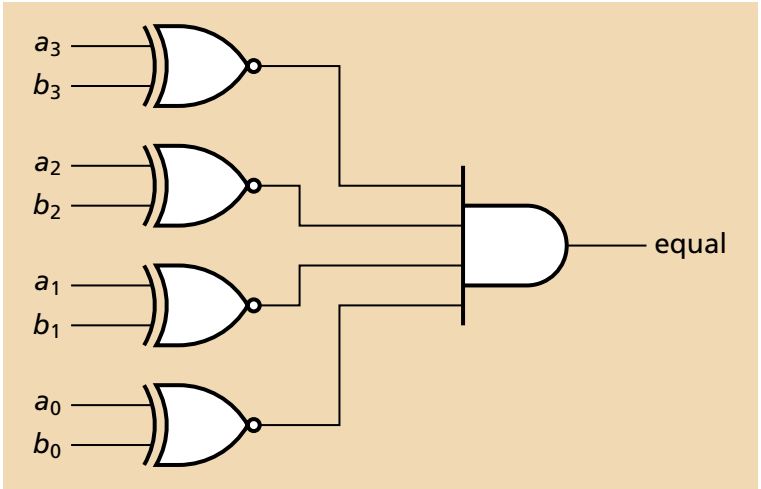
Logic Circuit to Compare Two 4-bit Binary Numbers



Logic Circuit to Compare Two 4-bit Binary Numbers



Logic Circuit to Compare Two 4-bit Binary Numbers



Euclid's Algorithm in Assembly Language

Assembly:

Python:

```
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
return a
```

Euclid's Algorithm in Assembly Language

Assembly:

Python:

```
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
return a
```

Assembly Registers:

$zero	0
$a0	"a"
$a1	"b"
$v0	result

Euclid's Algorithm in Assembly Language

Assembly:

```
subu $a0, $a0, $a1      # Subtract b from a
```

Python:

```
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
return a
```

Assembly Registers:

$zero	0
$a0	"a"
$a1	"b"
$v0	result

Euclid's Algorithm in Assembly Language

Assembly:

```
subu $a0, $a0, $a1     # Subtract b from a

subu $a1, $a1, $a0     # Subtract a from b
```

Python:

```
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
return a
```

Assembly Registers:

$zero	0
$a0	"a"
$a1	"b"
$v0	result

Euclid's Algorithm in Assembly Language

Assembly:

```
  sgt  $v0, $a1, $a0     # Is b > a?
  bne  $v0, $zero, .L1   # Yes, goto .L1
  subu $a0, $a0, $a1     # Subtract b from a

.L1:
  subu $a1, $a1, $a0     # Subtract a from b
```

Python:

```
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
return a
```

Assembly Registers:

$zero	0
$a0	"a"
$a1	"b"
$v0	result

Euclid's Algorithm in Assembly Language

Assembly:

```
gcd:
  beq  $a0, $a1, .L2     # if a = b, go to exit
  sgt  $v0, $a1, $a0     # Is b > a?
  bne  $v0, $zero, .L1   # Yes, goto .L1
  subu $a0, $a0, $a1     # Subtract b from a
  b    gcd               # and repeat
.L1:
  subu $a1, $a1, $a0     # Subtract a from b
  b    gcd               # and repeat
.L2:
```

Python:

```
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
return a
```

Assembly Registers:

$zero	0
$a0	"a"
$a1	"b"
$v0	result

Euclid's Algorithm in Assembly Language

Assembly:

```
gcd:
  beq  $a0, $a1, .L2     # if a = b, go to exit
  sgt  $v0, $a1, $a0     # Is b > a?
  bne  $v0, $zero, .L1   # Yes, goto .L1
  subu $a0, $a0, $a1     # Subtract b from a
  b    gcd               # and repeat
.L1:
  subu $a1, $a1, $a0     # Subtract a from b
  b    gcd               # and repeat
.L2:
  move $v0, $a0          # return a
  j    $ra               # Return to caller
```

Python:

```
while a != b:
  if a > b:
    a = a - b
  else:
    b = b - a
return a
```

Assembly Registers:

$zero	0
$a0	"a"
$a1	"b"
$v0	result

Euclid's Algorithm in Assembly and Machine Language

Assembly:

```
gcd:
  beq  $a0, $a1, .L2     # if a = b, go to exit
  sgt  $v0, $a1, $a0     # Is b > a?
  bne  $v0, $zero, .L1   # Yes, goto .L1
  subu $a0, $a0, $a1     # Subtract b from a
  b    gcd               # and repeat
.L1:
  subu $a1, $a1, $a0     # Subtract a from b
  b    gcd               # and repeat
.L2:
  move $v0, $a0          # return a
  j    $ra               # Return to caller
```

Machine Language:

```
00010000100001010000000000000111
00000000101001000001000000101010
00010100010000000000000000000011
00000000101001000010100000100011
00000100000000001111111111111100
00000000100001010010000000100011
00000100000000001111111111111010
00000000000000100000100000100001
00000011111100000000000000001000
```

Machine Language Instruction Encoding

```
subu $a0, $a0, $a1
```

Machine Language Instruction Encoding

subu `$a0, $a0, $a1`

opcode rd, rs, rt

subu encoding from the MIPS instruction set reference:

SPECIAL 000000	rs	rt	rd	0 00000	SUBU 100011

Machine Language Instruction Encoding

subu `$a0, $a0, $a1`

opcode rd, rs, rt

subu encoding from the MIPS instruction set reference:

SPECIAL 000000	rs	rt	rd	0 00000	SUBU 100011

$a0 is “00100” and $a1 is “00101”, so

000000	00100	00101	00100	00000	100011

Machine Language Instruction Encoding

subu `$a0, $a0, $a1`

opcode rd, rs, rt

subu encoding from the MIPS instruction set reference:

SPECIAL 000000	rs	rt	rd	0 00000	SUBU 100011

$a0 is “00100” and $a1 is “00101”, so

000000	00100	00101	00100	00000	100011

Glue bits together 00000000100001010010000000100011

Machine Language Instruction Encoding

subu `$a0, $a0, $a1`

opcode rd, rs, rt

subu encoding from the MIPS instruction set reference:

SPECIAL 000000	rs	rt	rd	0 00000	SUBU 100011

$a0 is “00100” and $a1 is “00101”, so

000000	00100	00101	00100	00000	100011

Break into groups of 4 0000 0000 1000 0101 0010 0000 0010 0011

Machine Language Instruction Encoding

subu `$a0, $a0, $a1`

opcode rd, rs, rt

subu encoding from the MIPS instruction set reference:

SPECIAL 000000	rs	rt	rd	0 00000	SUBU 100011

$a0 is “00100” and $a1 is “00101”, so

000000	00100	00101	00100	00000	100011

Break into groups of 4 0000 0000 1000 0101 0010 0000 0010 0011

Translate to Hexadecimal 0x00852023

Settings | File | Edit | Euclid's GCD | See Project Page | Tutorials | Debug | Save Now | not0my1real2name

Code | Costumes | Sounds

Motion
Looks
Sound
Events
Control
Sensing
Operators
Variables
My Blocks

mod
round
abs of

Variables

Make a Variable

a
b
my variable

set a to 0
change a by 1
show variable a
hide variable a

when clicked
ask What's the first number? and wait
set a to answer
ask What's the second number? and wait
set b to answer
repeat until a = b
if a > b then
set a to a - b
else
set b to b - a
say join The GCD is a for 2 seconds
stop all

What's the second number?

a 15
b 0

Sprite Sprite1 x 0 y 0
Show Size 100 Direction 90

Stage
Backdrops 1

Sprite1

(PRAXIS 5652: Computer Sciences Sample Test Questions)

4. Consider the following list.

- Assembly language
- Block-based programming language
- Logic gate
- Machine language

Which of the following arranges the list in order from highest level of abstraction to lowest level of abstraction?

A. Block-based programming language, assembly language, machine language, logic gate

B. Block-based programming language, machine language, assembly language, logic gate

C. Block-based programming language, machine language, logic gate, assembly language

D. Machine language, block-based programming language, assembly language, logic gate