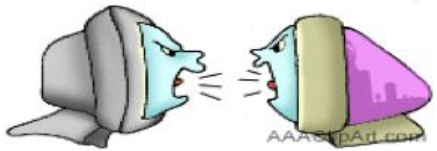
Microcode

CSC 236

CISC vs RISC



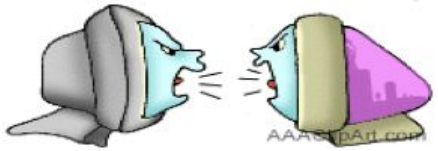
CISC - 1960 ... 1990 Microcoding

RISC - 1990 Hardwired

Complex Instruction Set Computing

Reduced Instruction Set Computing

CISC vs RISC



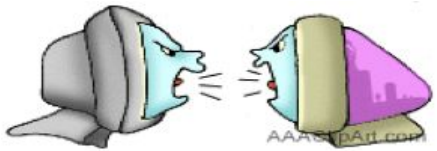
CISC - 1960 ... 1990 Microcoding

RISC - 1990 Hardwired

CISC - As chip costs rapidly decreased (1960-1990) engineers put more and more function into the hardware.

CISC vs RISC

- **Lots of instructions**
- **Complicated instruction decode**
- **Slow / Variable instruction execution**



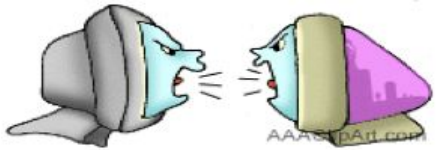
CISC - 1960 ... 1990 Microcoding

RISC - 1990 Hardwired

- **Many ways to access data (address modes)**
- **Machine Instructions**
  - **complicated and fancy**
  - **variable size**
  - **variable execution time**

CISC vs RISC

- **Lots of instructions**
- **Complicated instruction decode**
- **Slow / Variable instruction execution**



CISC - 1960 ... 1990
Microcoding

RISC - 1990
Hardwired

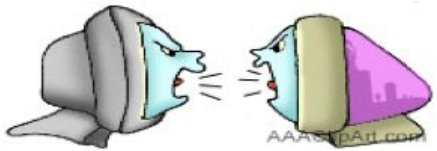
- **Many ways to access data (addressing modes)**
- **Machine instructions**
  - **complex and fancy**
  - **variable size**
  - **variable execution time**

Too complex
Too specific

Circa 1990 … examination of real code showed programmers were not using the complex functions

CISC vs RISC

- **Lots of instructions**
- **Complicated instruction decode**
- **Slow / Variable instruction execution**



CISC - 1960 ... 1990
Microcoding

RISC - 1990
Hardwired

- **Lean hardware**
- **Simple decode**
- **Fast**

- **Many ways to access data les)**
- **Mach ions**
  - **complicated and fancy**
  - **variable size**
  - **variable execution time**

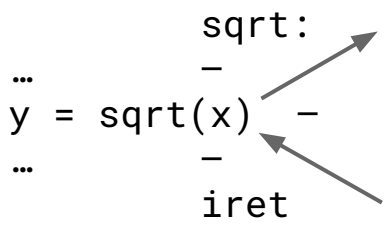
Intel
x86

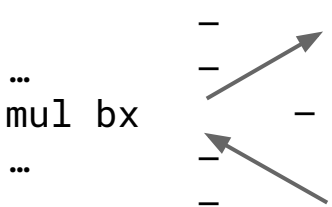
- **Only simple load and sto ory**
- **Ma**
  Ins
  - **only basic operations**
  - **fixed size**
  - **fixed execution times**

ARM
Advanced
RISC Machine

Microcode

- Implements machine instructions
  - Using sub-instructions — micro-instructions
  - Not visible off chip
  - Not visible to assembly language programmer
  - Each visible machine instruction is performed by 1+ micro-instruction
- Analogous to a subroutine call





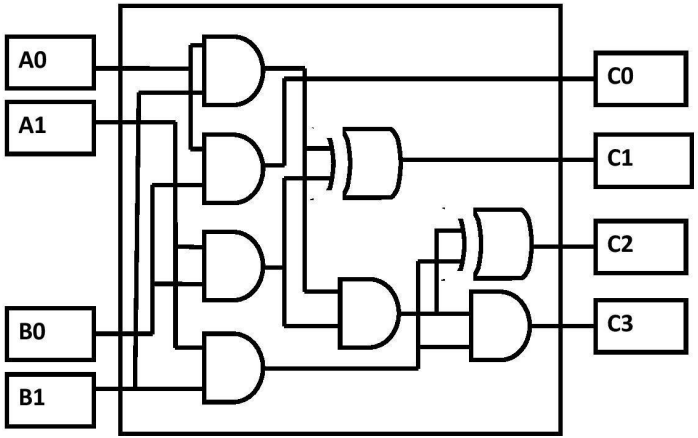
Tradeoffs microcode

- Benefits
  - Increased functionality
  - Increased flexibility
  - Reduced hardware complexity
  - Performance*
- Costs
  - Performance*
  - Lack of uniformity

(*) performance is a 2-way street

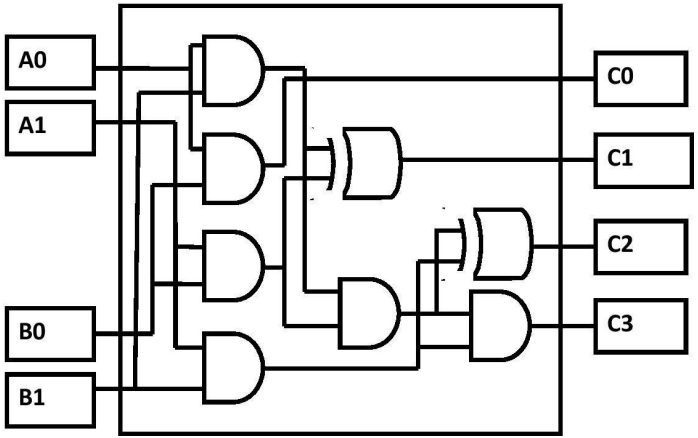
- **All operations will be slower than direct implementation in hardware.**
- **A single complex instruction can do the job of many simple instructions.**

2 bit x 2 bit binary multiplier



A1 A0 x B1 B0 = C3 C2 C1 C0

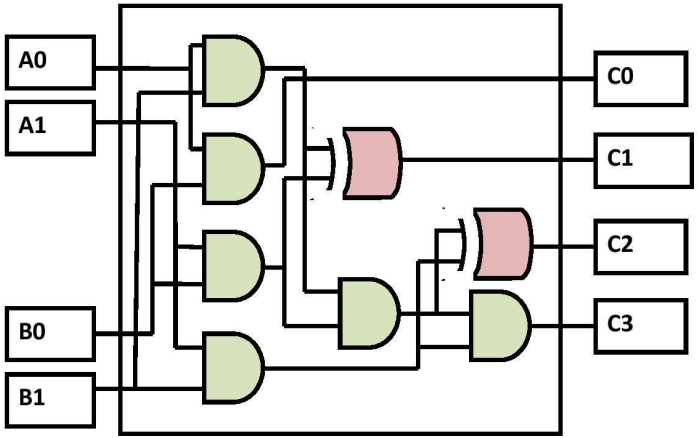
2 bit x 2 bit binary multiplier



A1 A0 x B1 B0 = C3 C2 C1 C0

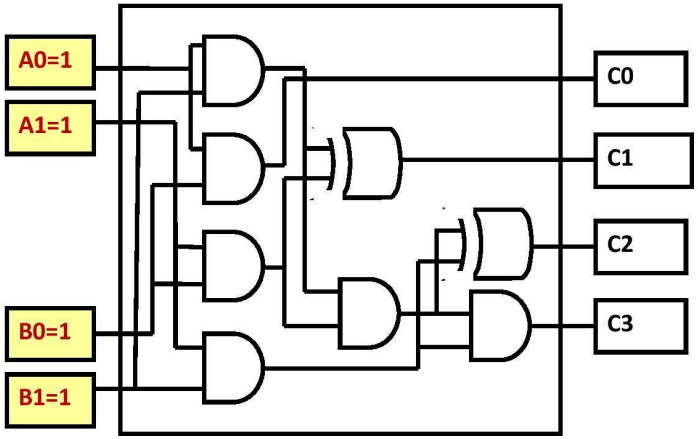
1 1 x 1 1 = 1 0 0 1

2 bit x 2 bit binary multiplier



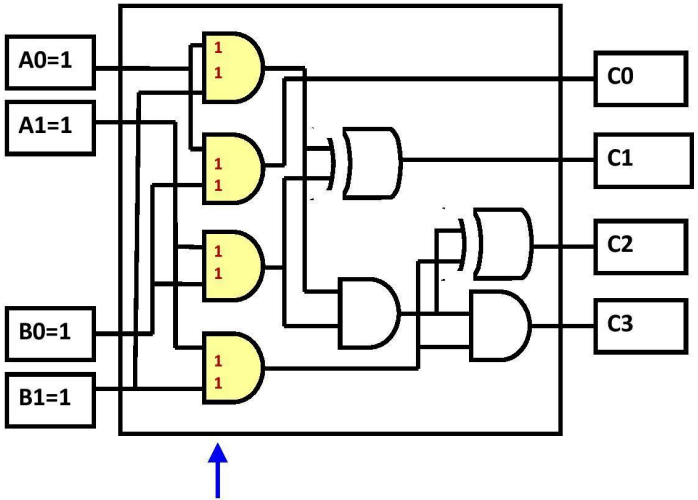
6 *AND* gates **2 *XOR* gates**

2 bit x 2 bit binary multiplier



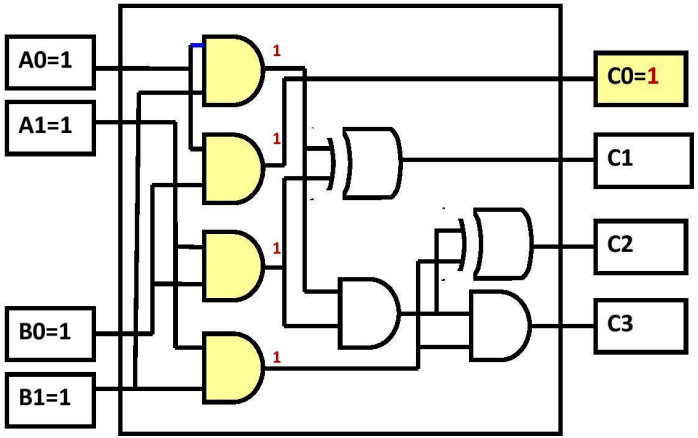
$11_2 \times 11_2 = ? = 1001_2$

2 bit x 2 bit binary multiplier



Input to these
***and* gates will be 1**

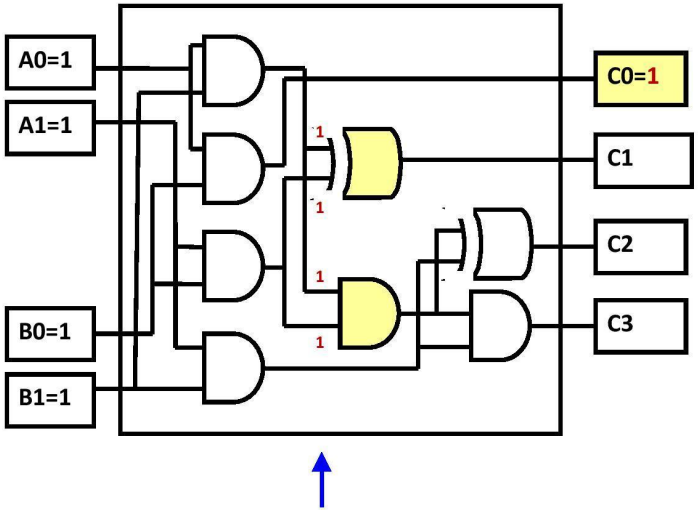
2 bit x 2 bit binary multiplier



Output of these *and* gates will be 1

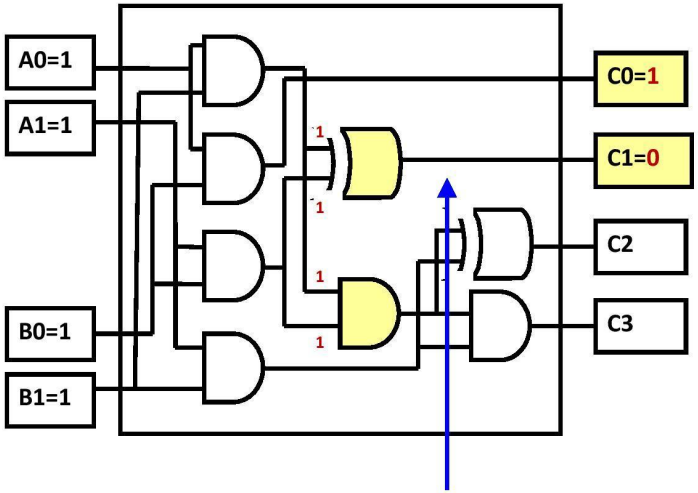
(one gate feeds C0)

2 bit x 2 bit binary multiplier



Input to these gates will be 1

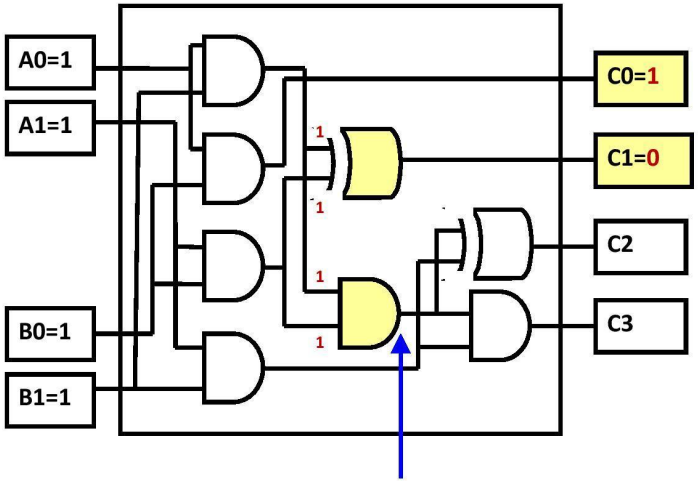
2 bit x 2 bit binary multiplier



Output of this *xor* gate will be 0

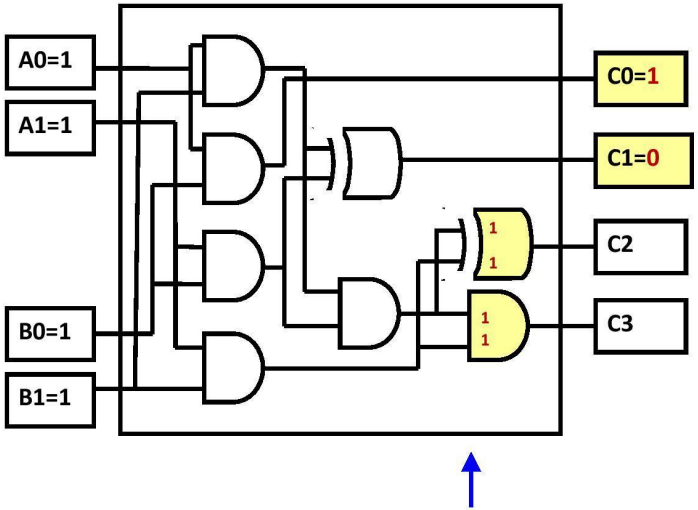
(feeds C1)

2 bit x 2 bit binary multiplier



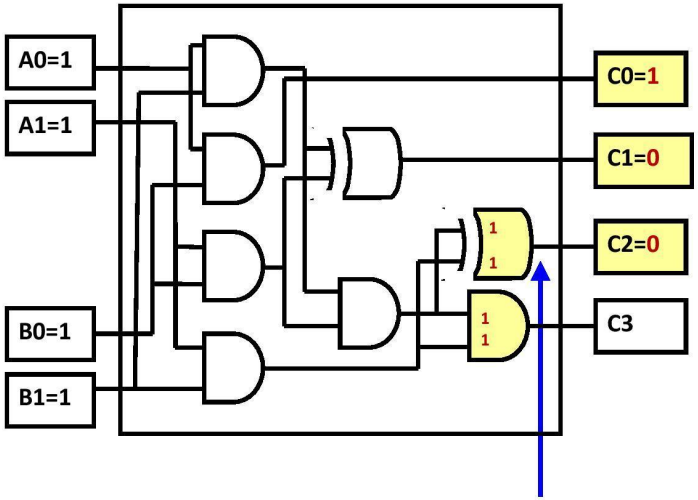
Output of this *and* gate will be 1

2 bit x 2 bit binary multiplier



Input to these gates will be 1

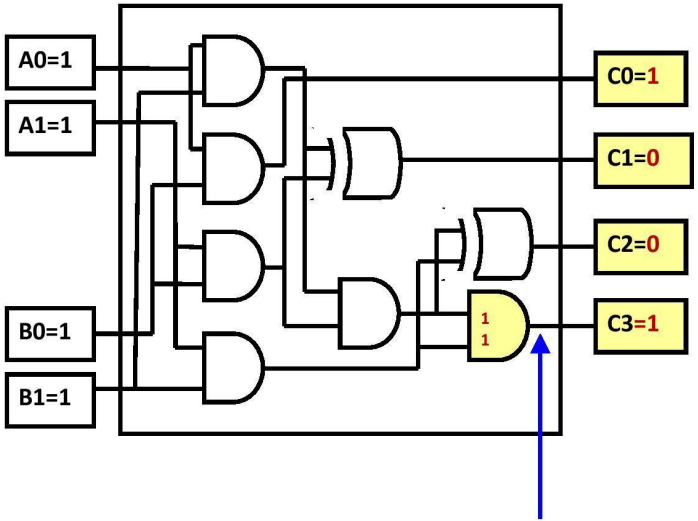
2 bit x 2 bit binary multiplier



Output will be 0

(feeds C2)

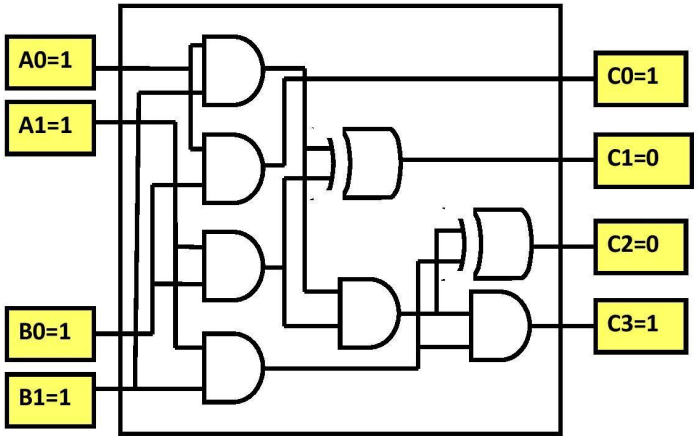
2 bit x 2 bit binary multiplier



Output will be 1

(feeds C3)

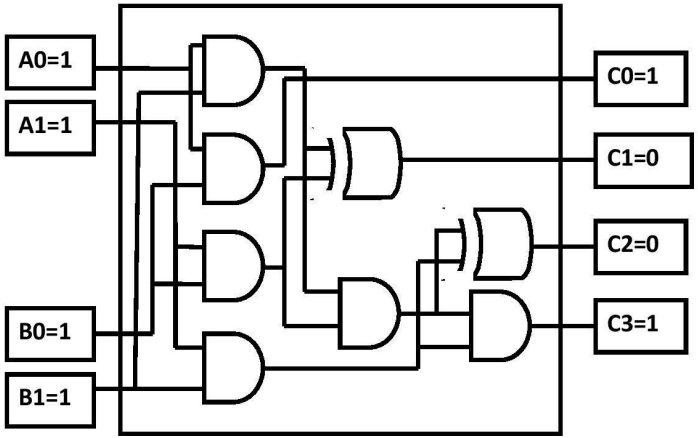
2 bit x 2 bit binary multiplier



$11_2 \times 11_2 = 1001_2$

$3_{10} \times 3_{10} = 9_{10}$

2 bit x 2 bit binary multiplier

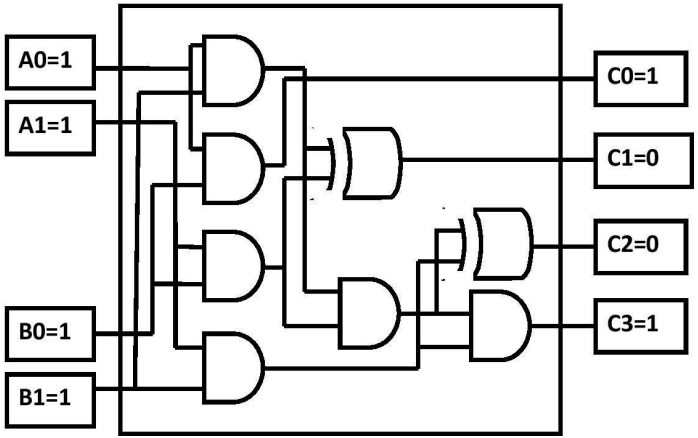


This is an example of a

hardwired instruction

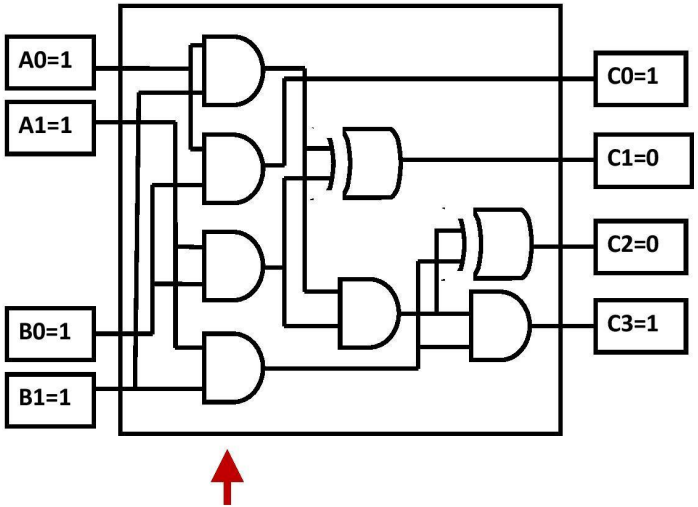
handcrafted from logic gates

2 bit x 2 bit binary multiplier



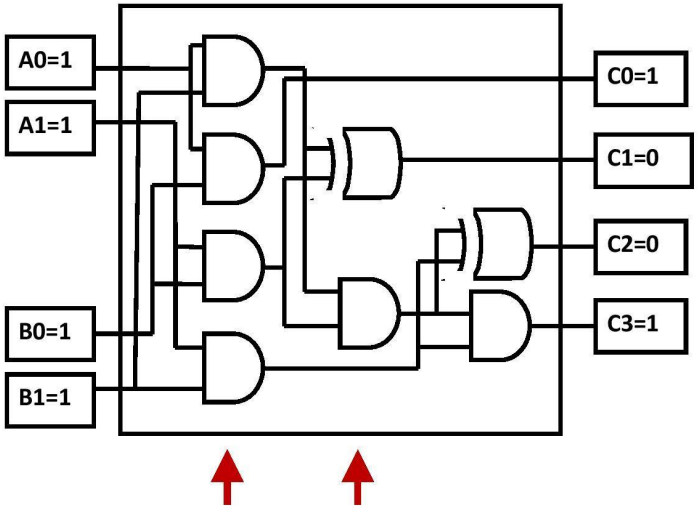
The *time* to execute depends on the number of gates the signal must propagate through

2 bit x 2 bit binary multiplier



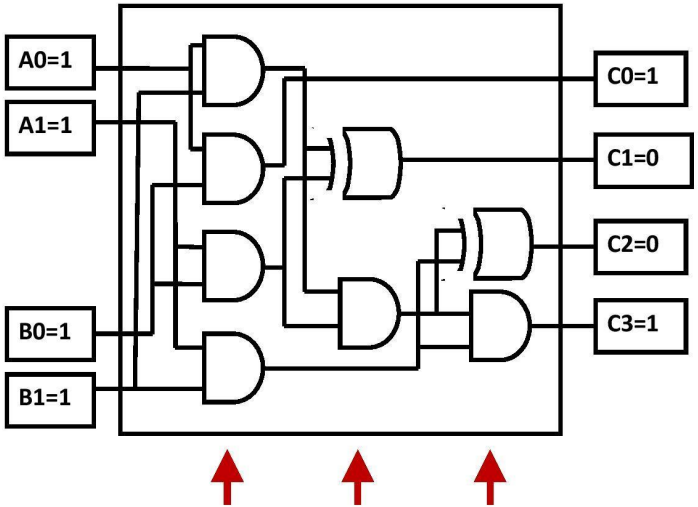
The *time* to execute depends on the number of gates the signal must propagate through

2 bit x 2 bit binary multiplier



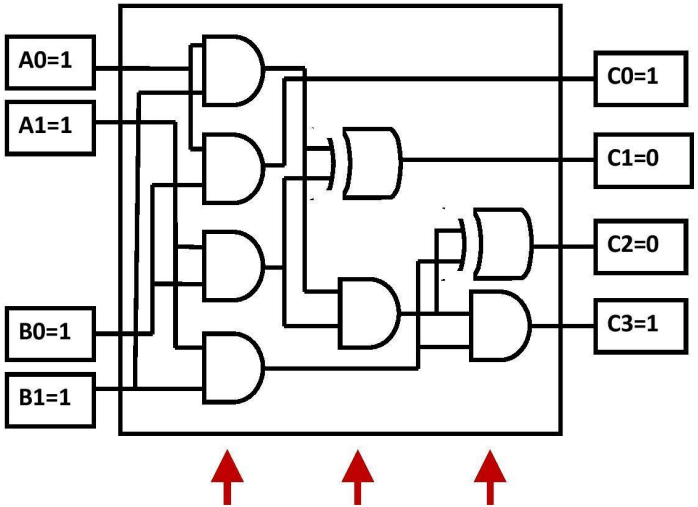
The *time* to execute depends on the number of gates the signal must propagate through

2 bit x 2 bit binary multiplier



The *time* to execute depends on the number of gates the signal must propagate through

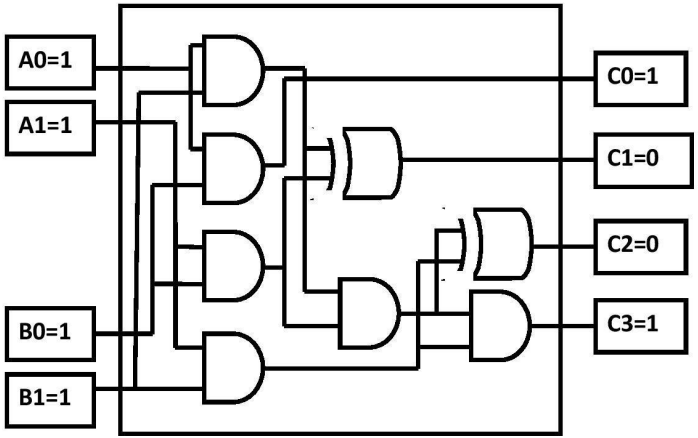
2 bit x 2 bit binary multiplier



Gate Delay

the number of gates the signal must propagate through

2 bit x 2 bit binary multiplier



2 x 2 multiply

8 gates

This is easy

A 16-bit multiply would be … harder.

Advantages of microcode

- Cost
  - Cheaper to provide same functionality
- Example
  - 8086 — 29,000 transistors
  - 16x16 bit multiply requires 1000s of transistors
  - Microcode performs multiply
    - Using shifts and adds in a loop
    - Variable execution time
    - 118-133 cycles for 16x16 bit multiply
    - 144-162 cycles for 32/16 bit divide

Pentium

- **42,000,000 transistors**
- **11 cycles for 16x16 bit multiply**
- **25 cycles for 32/16 bit divide**

Shifts and adds

A = 0011
B = 0011 **C =** 0000

n = 0

if nth bit of B == 1
add A
shift A left
n++

Shifts and adds

```
          0011
0011      0000
```

```
0
```

if nth bit of B == 1
**   add A**
shift A left
n++

Shifts and adds

```
          0011
0011      0011
```

```
0
```

if nth bit of B == 1
add A
shift A left
n++

Shifts and adds

```
          0110
0011      0011
```

```
0         if nth bit of B == 1
                add A
          shift A left
          n++
```

Shifts and adds

```
          0110
0011      0011
```

1

if nth bit of B == 1
add A
shift A left
n++

Shifts and adds

```
          0110
0011      1001
```

1

if nth bit of B == 1
add A
shift A left
n++

Shifts and adds

```
          1100
0011      1001
```

1

if nth bit of B == 1
add A
shift A left
n++

Shifts and adds

```
        1100
0011    1001
```

2

```
if nth bit of B == 1
        add A
shift A left
n++
```

Shifts and adds

```
        1100
0011    1001
```

2

if nth bit of B == 1
add A
shift A left
n++

Shifts and adds

```
          1000
0011      1001
```

3

if nth bit of B == 1
add A
shift A left
n++

Shifts and adds

```
        1000
0011    1001
```

3

if nth bit of B == 1
add A
shift A left
n++

Shifts and adds

```
          1000
0011      1001 = 9
```

3

if nth bit of B == 1
add A
shift A left
n++

Another Example

```
          0010
0111      0000
```

```
0         if nth bit of B == 1
                add A
          shift A left
          n++
```

Shifts and adds

```
          0010
0111      0010
```

```
0         if nth bit of B == 1
                add A
          shift A left
          n++
```

Shifts and adds

```
          0100
0111      0110
```

1

```
if nth bit of B == 1
        add A
shift A left
n++
```

Shifts and adds

```
          1000
0111      1110
```

= 14

2

```
if nth bit of B == 1
        add A
shift A left
n++
```

Shifts and adds

```
        1000
0111    1110        = 14
```

```
2       if nth bit of B == 1
            add A
        shift A left
        n++
```

```
0111 x 0010

  0000
+ 0010   =   1 x 0010 x 2^0
+ 0100   =   1 x 0010 x 2^1
+ 1000   =   1 x 0010 x 2^2
+ 0000   =   0 x 0010 x 2^3
```

Advantages of microcode

- Flexibility
  - Same microcode engine can support different ISA
- Example
  - IBM system 360 (circa 1960)
  - Could *emulate* many different architectures

Advantages of microcode

- Performance
  - One complex instruction instead of many simple ones
  - One long running instruction

    vs.

    Lots of short-running instructions
- Example — movsb
  - One machine instruction
    - Executing a handful of micro-operation per byte
  - Compared to writing your own loop
    - Executing a handful of machine instructions per byte
- Consider standard library functions like memcpy()
  - Could take advantage of special machine instructions.

Move string example

- Move 80 chars at a to b

```
    mov si, offset a        ;point to a
    mov di, offset b        ;point to b
    mov cx, 80          ;count is 80

do: mov al, [si]        ;get a char
    mov [di], al        ;put a char
    inc si          ;adv source ptr
    inc di          ;adv dest ptr
    dec cx          ;count = count - 1
    jne do          ;loop if more data
```

- **Static instructions: 6 vs. 1**
- **Dynamic instructions:**
  - **80 x 6 vs. 80 x 1**
  - **480 vs. 80**
- **How much faster:**
  - **Less than 6x?**
  - **More than 6x?**

```
rep movsb
```