

## How Computers Work(cont.)

Jun 8, 2007  
KangKang Yin

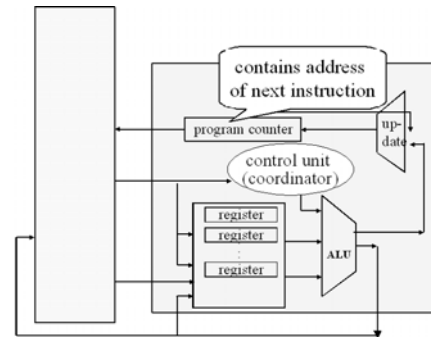
## Basic Assembly Instructions

- ▶ **add, subtract, multiply**: data stored in the processor
- ▶ **load, store**: get data from memory to registers and back
- ▶ **branch, jump**: instructions for control flow

## example instructions

- ▶ **add 3, 4, 3**
  - add the contents of registers 3 and 4, and store the answer in register 3
- ▶ **load 3, 7000**
  - load into register 3 the contents of memory location 7000
- ▶ **jmp 7:**
  - move ahead by 7 instructions

## Processor organization



## the program counter

- ▶ the control unit retrieves the next instruction from memory
- ▶ contains the memory address of the next instruction
- ▶ points to the next instruction automatically by the circuitry after each instruction is fetched

## Fetch/Execute Cycle

### Instruction Fetch

- 1 fetch instruction specified by program counter

### Instruction Decode

- 2 decode instruction

### Data Fetch

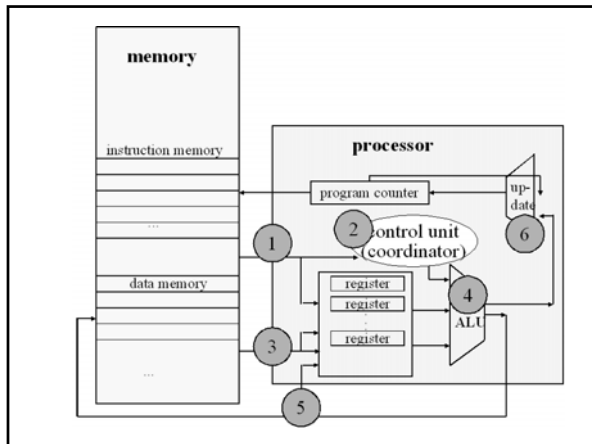
- 3 fetch data from memory and store in registers

### Instruction Execution

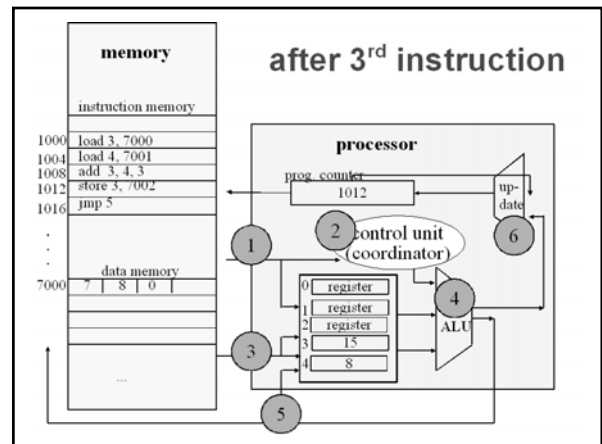
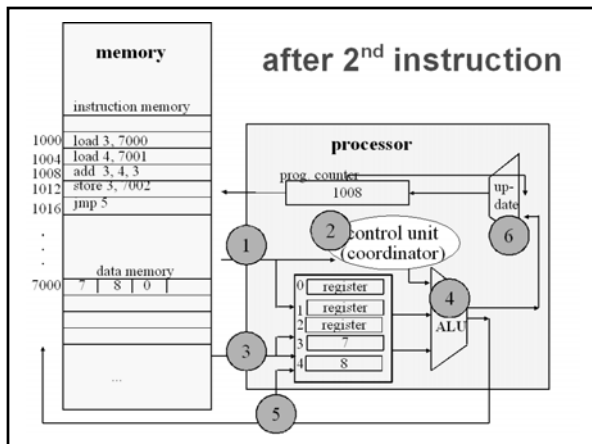
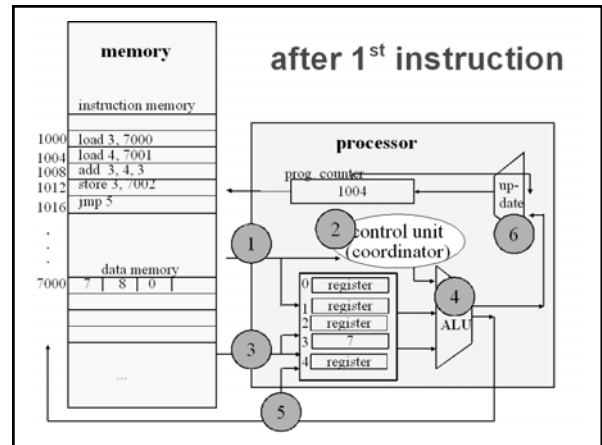
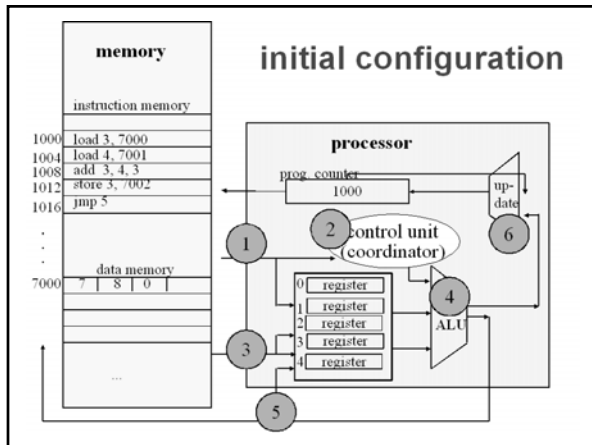
- 4 perform operation

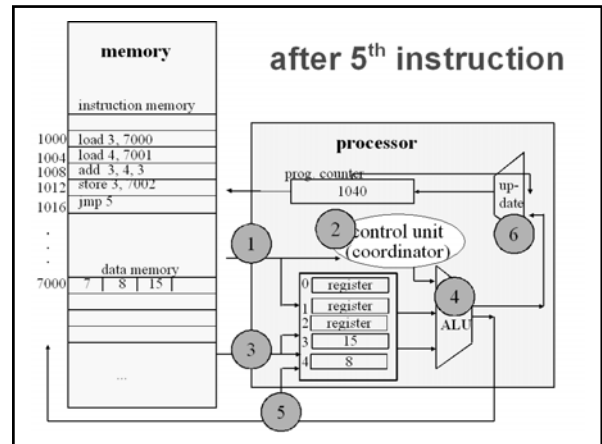
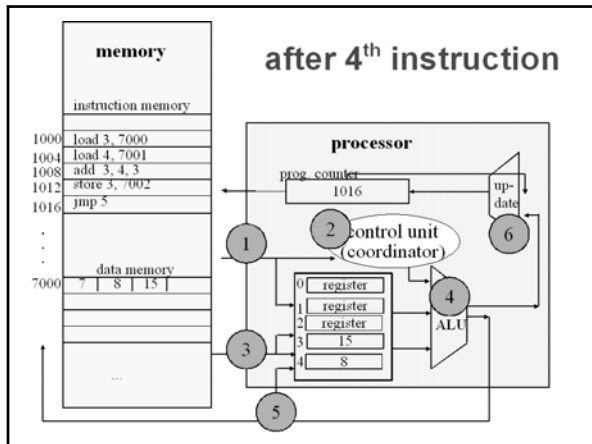
### Result Return

- 5 to data memory
- 5 to a register
- 6 update program counter



- ### putting it all together
- ▶ **load 3, 7000:** load into register 3 the contents of memory location 7000
  - ▶ **load 4, 7001:** load into register 4 the contents of memory location 7001
  - ▶ **add 3, 4, 3:** add the contents of registers 3 and 4 and store in register 3
  - ▶ **store 3, 7002:** store the contents of register 3 in memory location 7002
  - ▶ **jmp 5:** move ahead 5 instructions





### Clock speed

- ▶ clock: a series of pulses generated by an oscillator that sets the tempo for the processor
- ▶ 1 gigahertz: 1 cycle per nanosecond
- ▶ size-speed relationship

### Faster and smaller

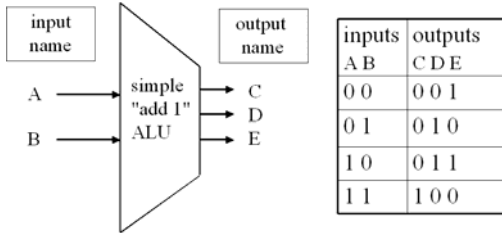
- ▶ increasing processing speed:
  - multiple processors
  - caches (stores of data on the processor)
  - pipelining (starting one instruction before the last one finishes)
- ▶ Moore's law: <http://www.physics.udel.edu/~watson/scen103/intel.html>

### Step3: from electrons to logic

### An ALU close-up

- performs an *operation* on the data
- Arithmetic: addition; subtraction; multiplication; division
  - Boolean algebra: Or; AND; Not

design a unit that  
adds 1 to a 2-bit input

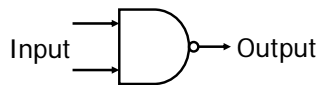


## Logic Gates

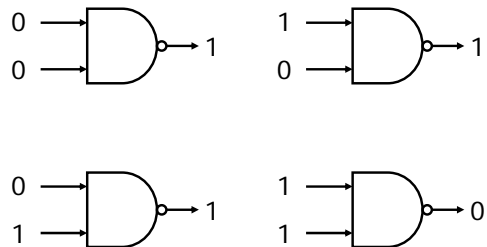
- Basic elements of digital circuits
- [http://en.wikipedia.org/wiki/Logic\\_gates](http://en.wikipedia.org/wiki/Logic_gates)

## NAND Gate

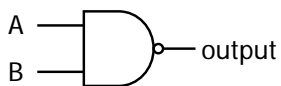
► <http://www.fairchildsemi.com/>



## NAND Input/Output

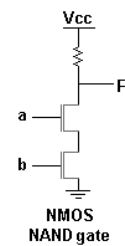


## NAND Truth Table

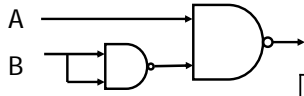


A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

## The Amazing Transistor

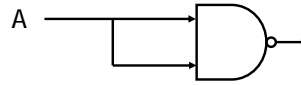


### gates → circuits



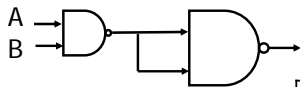
A	B	output
0	0	1
0	1	1
1	0	0
1	1	1

### More circuits(negator)



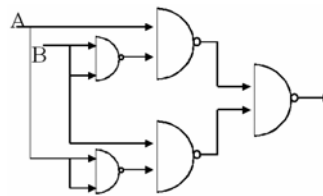
A	output
0	1
1	0

### More circuits(AND)



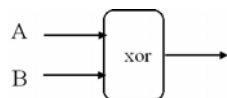
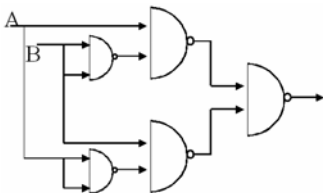
A	B	output
0	0	0
0	1	0
1	0	0
1	1	1

### XOR: exclusive OR

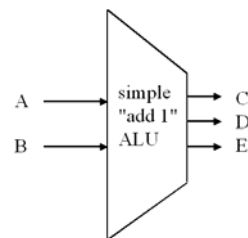


A	B	output
0	0	0
0	1	1
1	0	1
1	1	0

### XOR abstraction

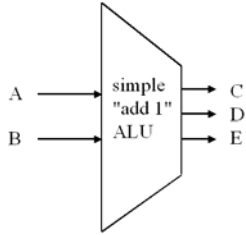


### Back to our task



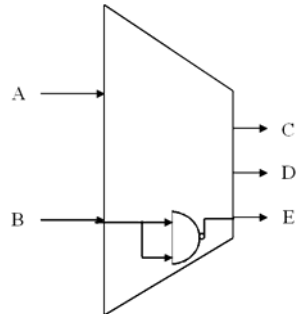
inputs	outputs
AB	CDE
00	001
01	010
10	011
11	100

first focus on output E



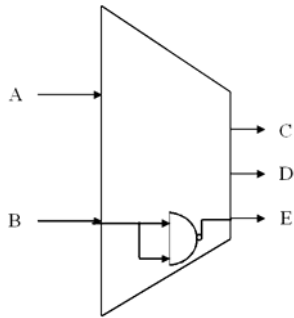
inputs		outputs	
AB	CDE		
00	1		
01	0		
10	1		
11	0		

$E = \sim B$



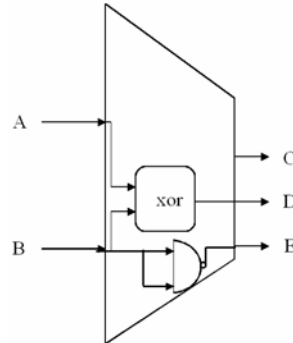
inputs		outputs	
AB	CDE		
00	0		
01	1		
10	1		
11	0		

then look at output D



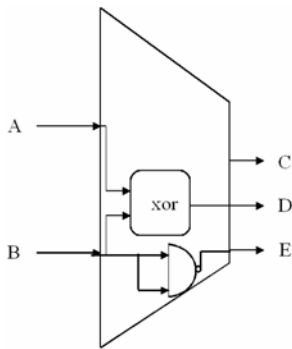
inputs		outputs	
AB	CDE		
00	0		
01	1		
10	1		
11	0		

$D = A \wedge B$



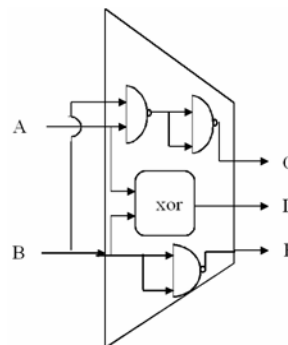
inputs		outputs	
AB	CDE		
00	001		
01	010		
10	011		
11	100		

finally look at output C



inputs		outputs	
AB	CDE		
00	001		
01	010		
10	011		
11	100		

$C = A \& B$



inputs		outputs	
AB	CDE		
00	001		
01	010		
10	011		
11	100		

## Summary

- ▶ NAND
  - the easiest to manufacture,
  - functional completeness: any other logic function (AND, OR, etc.) can be implemented using only NAND gates.
  
- ▶ Under a dollar from major semiconductor manufacturers:
  - Fairchild Semiconductor
  - Philips
  - Texas Instruments