Ceng 111 – Fall 2018
Week 3

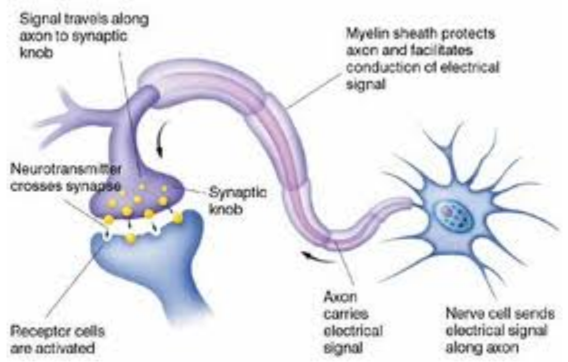
Computer Organization

Credit: Some slides are from the “Invitation to Computer Science” book by G. M. Schneider, J. L. Gersting and some from the “Digital Design” book by M. M. Mano and M. D. Ciletti.

Previously on CENG111

Computation in our brain

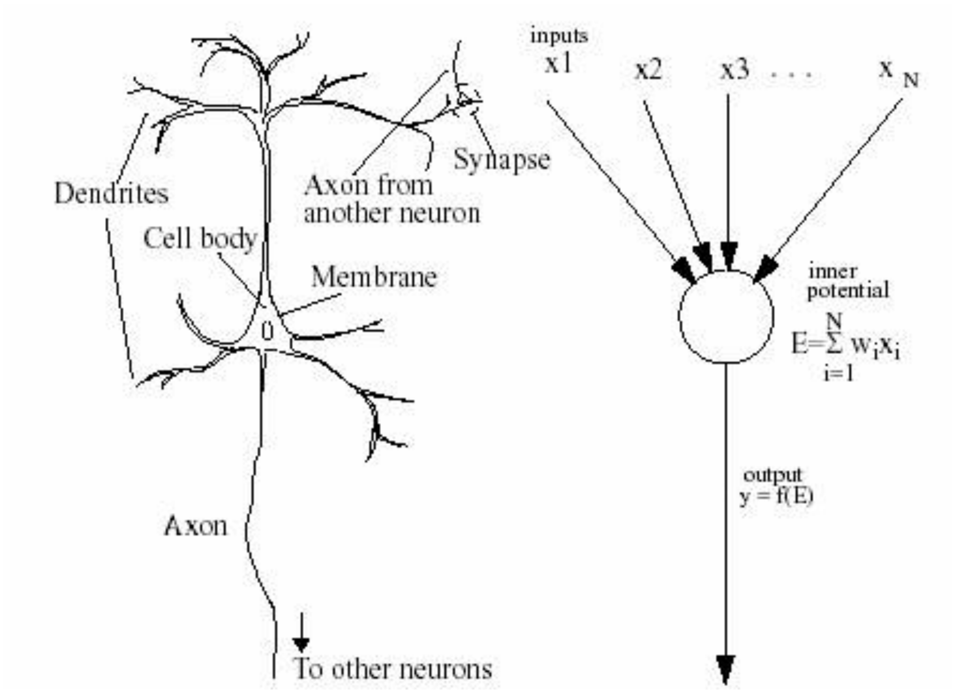
- Highly-connected network of neurons.
- How many neurons?
  - Approx. 10^{11} neurons and 10^{14} synapses.
- How do they transmit information?
  - Using nothing else than charged molecules.

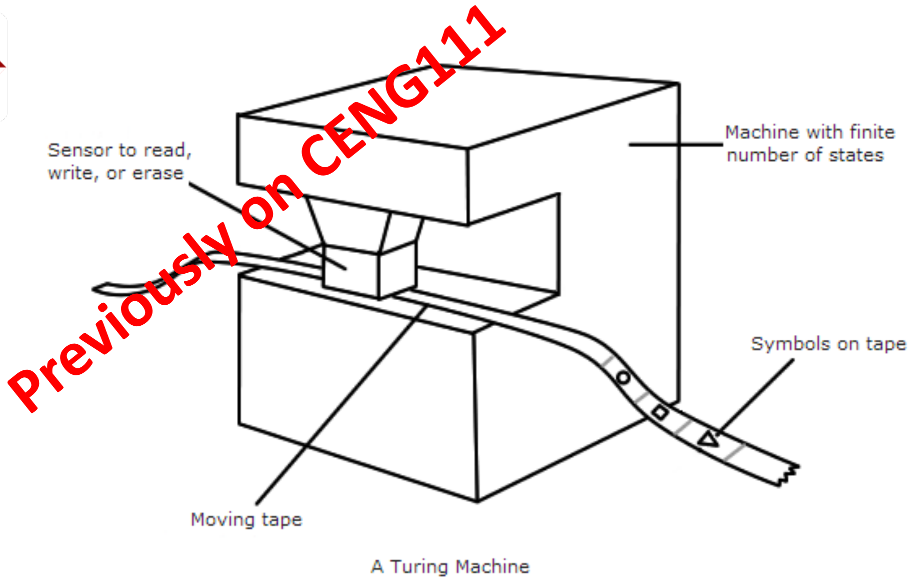

Signal travels along axon to synaptic knob
Myelin sheath protects axon and facilitates conduction of electrical signal
Neurotransmitter crosses synapse
Synaptic knob
Axon carries electrical signal
Receptor cells are activated
Nerve cell sends electrical signal along axon

Previously on CENG111

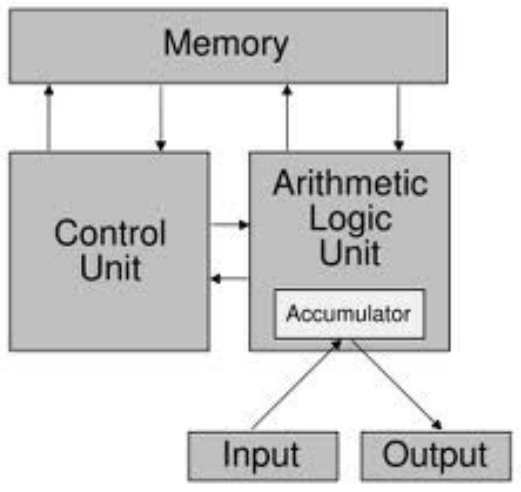
Computation in our brain (cont'd)

- Each neuron gets input and produces an output using an "activation function"




Previously on CENG111
Sensor to read, write, or erase
Machine with finite number of states
Symbols on tape
Moving tape
A Turing Machine

Turing Machine



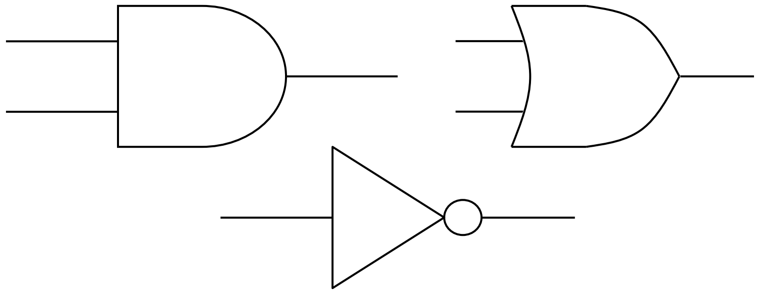
Von Neumann Architecture

DIGITAL COMPUTATION

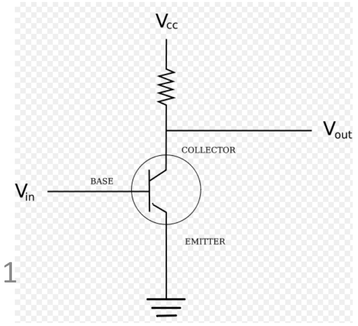
A computer

Previously on CENG111

Devices



Gates

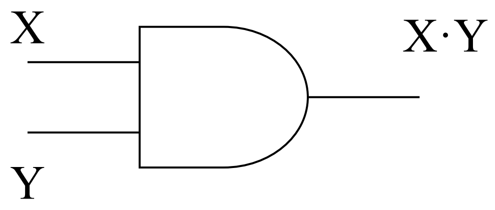


Transistors

Previously on CENG111

AND gate

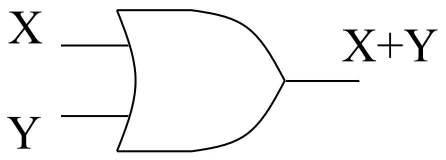
X	Y	$X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1



Previously on CENG111

OR Gate

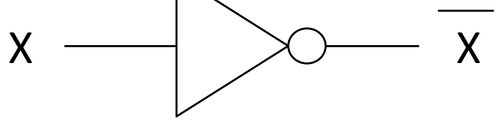
X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1



Previously on CENG111

NOT Gate

X	$\overline{X}$
0	1
1	0



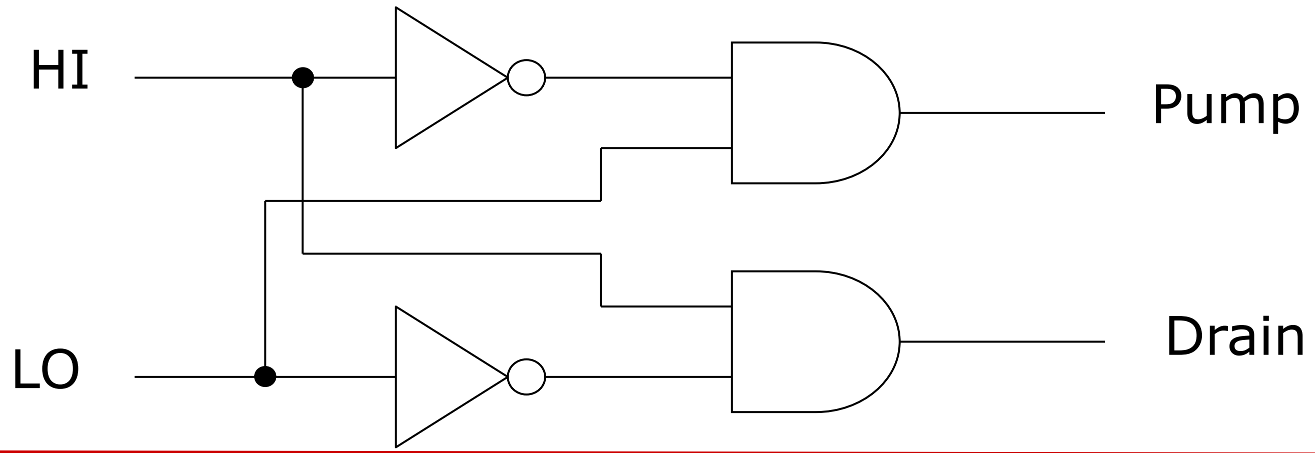
Previously on CENG111

An example problem: Water Tank

Truth Table Representation

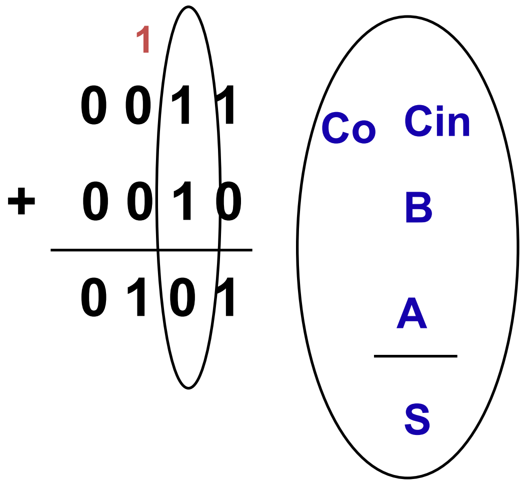
HI	LO	Pump	Drain	
0	0	0	0	Tank level is OK
0	1	**1**	0	Low level, pump more in
1	0	0	**1**	High level, drain some out
1	1	x	x	Inputs cannot occur

Schematic Representation



Previously on CENG111

1-bit full-adder

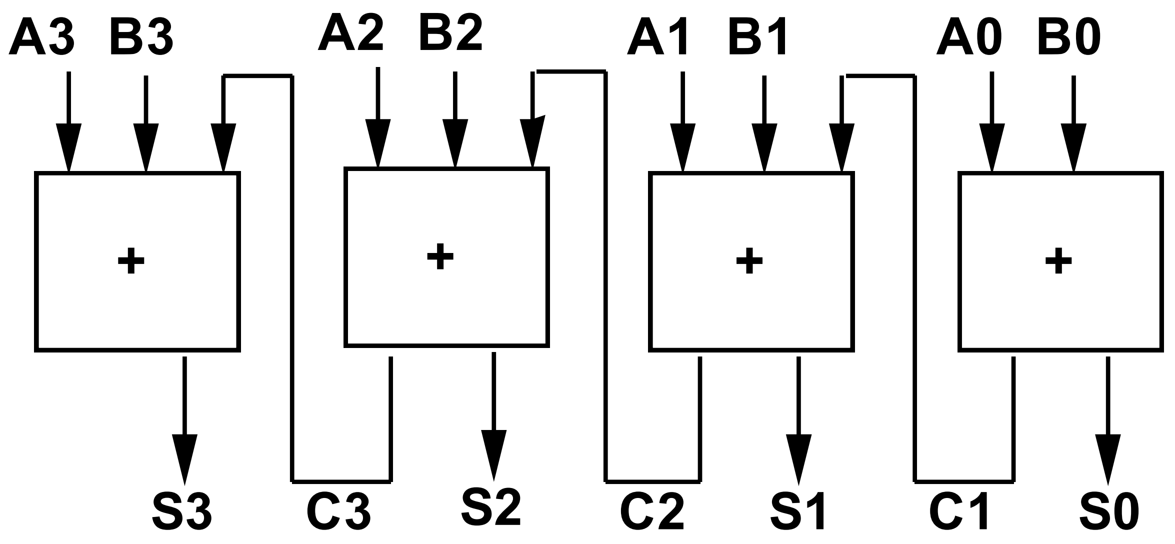


A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

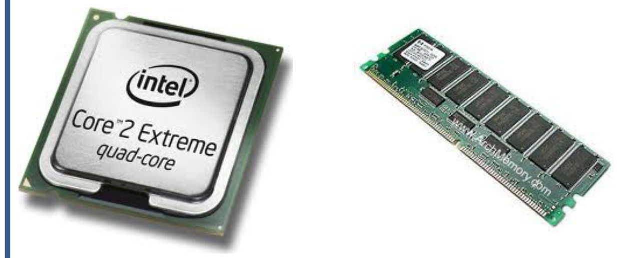
METU Computer Engineering

Previously on CENG111

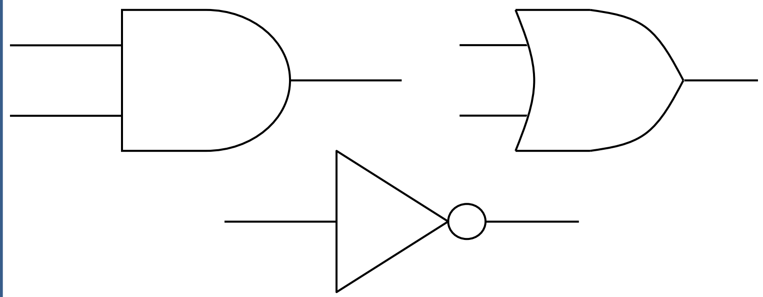
N-bit Adder



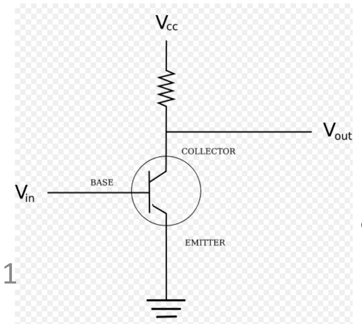
Today



Devices



Gates



Transistors

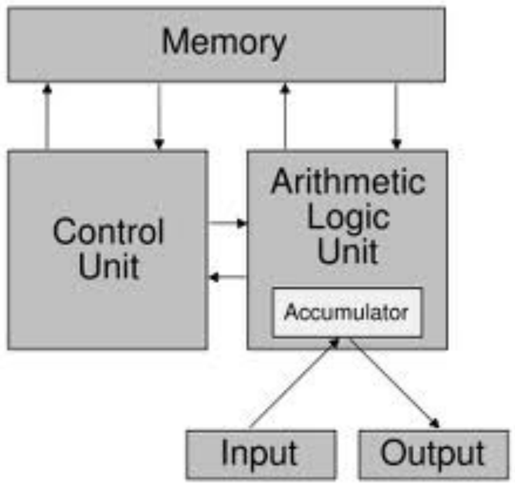
Today

- Computer Organization
  - CPU
  - Memory
  - Fetch-decode-execute cycle

METU Computer Engineering

Administrative Issues

- Busy hours for lab schedule
- Quiz



Computer Organization

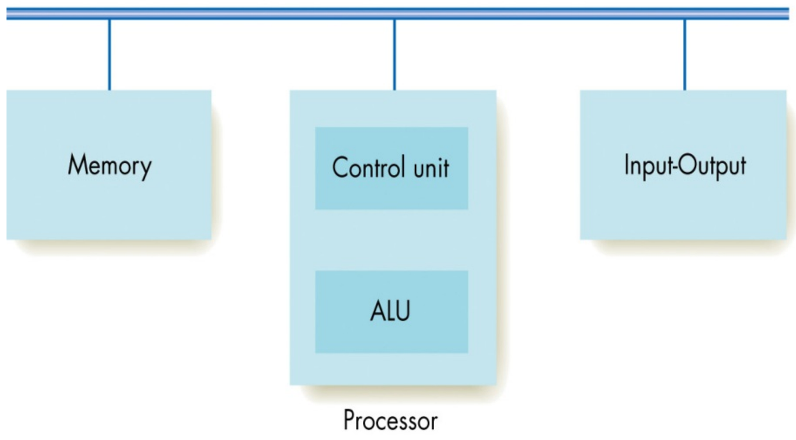
VON NEUMANN ARCHITECTURE & ITS IMPLEMENTATION

Von Neumann Architecture & Its Implementation

- How are instructions coded?
- How are instructions executed?
- How do the different subcomponents interact?
  - Memory
  - ALU
  - The Bus System
  - Registers

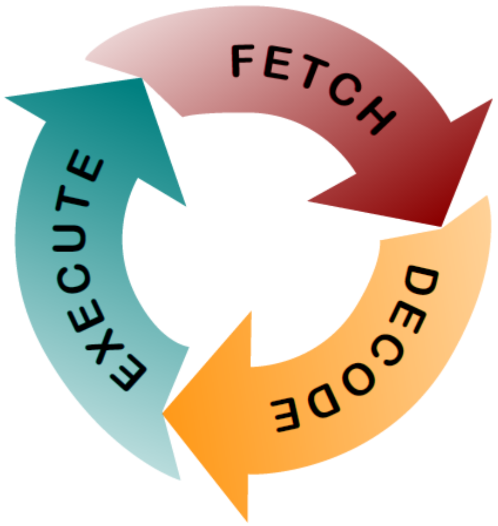
The Components of a Computer System

- Von Neumann architecture has four functional units:
  - Memory
  - Input/Output
  - Arithmetic/Logic unit
  - Control unit
- Sequential execution of instructions
- Stored program concept



METU Computer Engineering

Instruction Execution

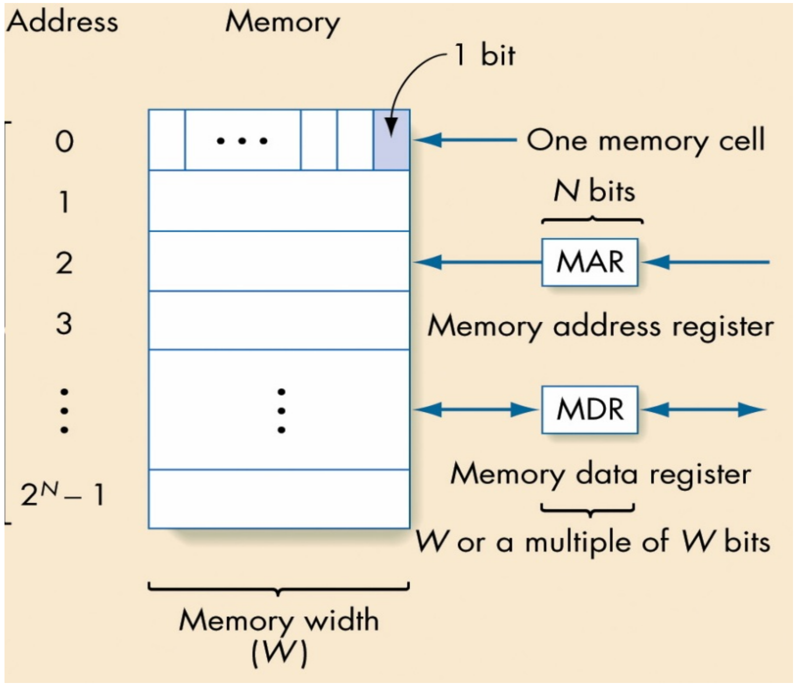


Memory and Cache

- Information is stored and fetched from memory subsystem
- Memory maps addresses to memory locations
- Cache memory keeps values currently in use in faster memory to speed access times

Memory and Cache (continued)

- RAM (Random Access Memory)
  Often called *memory*, *primary memory*
  - Memory made of addressable "cells"
  - Cell size is 8 bits
    - Nowadays, it is 32 or 64 bits.
  - All memory cells accessed in equal time
  - Memory address
    - Unsigned binary number with N bits
    - Address space is then 2^N cells



Memory and Cache (continued)

- Rapid access, low capacity “warehouse”
- Retains information entered through input unit
- Retains info that has already been processed until can be sent to output unit

Memory and Cache (continued)

- Parts of the memory subsystem
  - Fetch/store (or Read/Write) controller
    - Fetch: retrieve a value from memory
    - Store: store a value into memory
  - Memory address register (MAR)
  - Memory data register (MDR)

Memory and Cache (continued)

- Fetch operation
  - The address of the desired memory cell is moved into the MAR
  - Fetch/store controller signals a “fetch,” accessing the memory cell
  - The value at the MAR’s location flows into the MDR

METU Computer Engineering

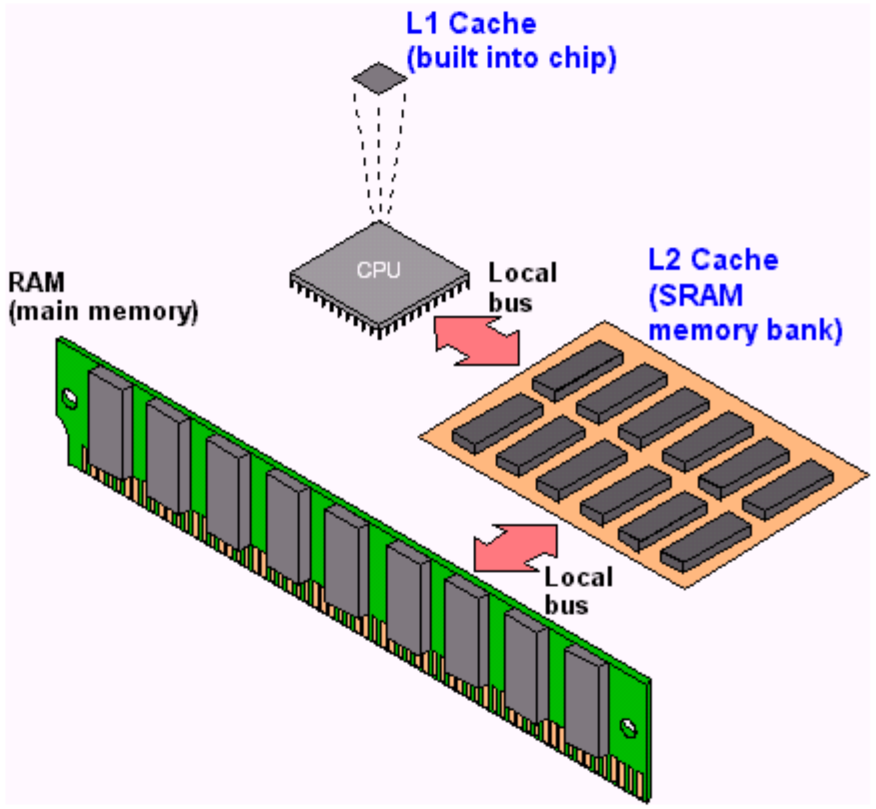
Memory and Cache (continued)

- Store operation
  - The address of the cell where the value should go is placed in the MAR
  - The new value is placed in the MDR
  - Fetch/store controller signals a “store,” copying the MDR’s value into the desired cell

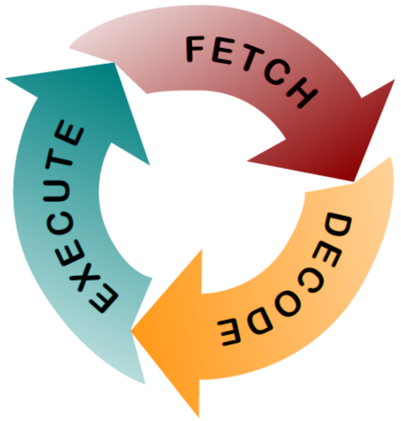
Cache Memory

- Memory access is much slower than processing time
- Faster memory is too expensive to use for all memory cells
- Locality principle
  - Once a value is used, it is likely to be used again
- Small size, fast memory just for values currently in use speeds computing time

From Computer Desktop Encyclopedia
© 1999 The Computer Language Co. Inc.



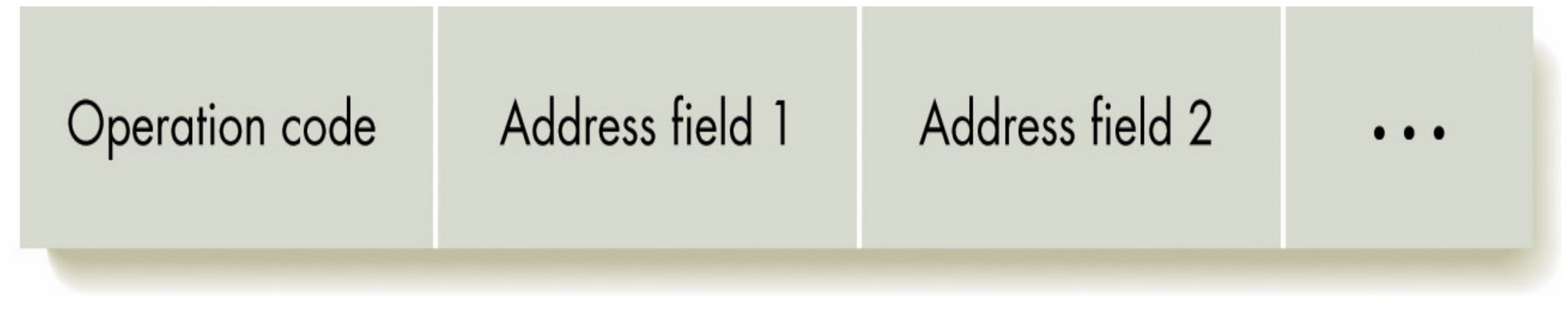
The Control Unit



- Manages stored program execution
- Task
  1. Fetch from memory the next instruction to be executed
  2. Decode it: determine what is to be done
  3. Execute it: issue appropriate command to ALU, memory, and I/O controllers

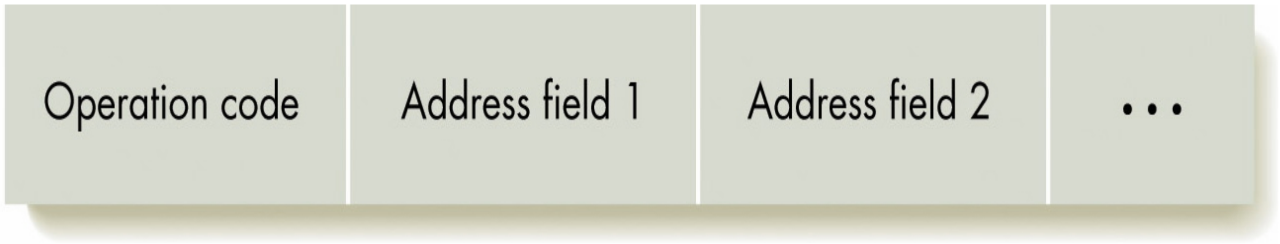
Machine Language Instructions

- Can be decoded and executed by control unit
- Parts of instructions
  - Operation code (op code)
    - Unique unsigned-integer code assigned to each machine language operation
  - Address field(s)
    - Memory addresses of the values on which operation will work



Typical Machine Language Instruction Format

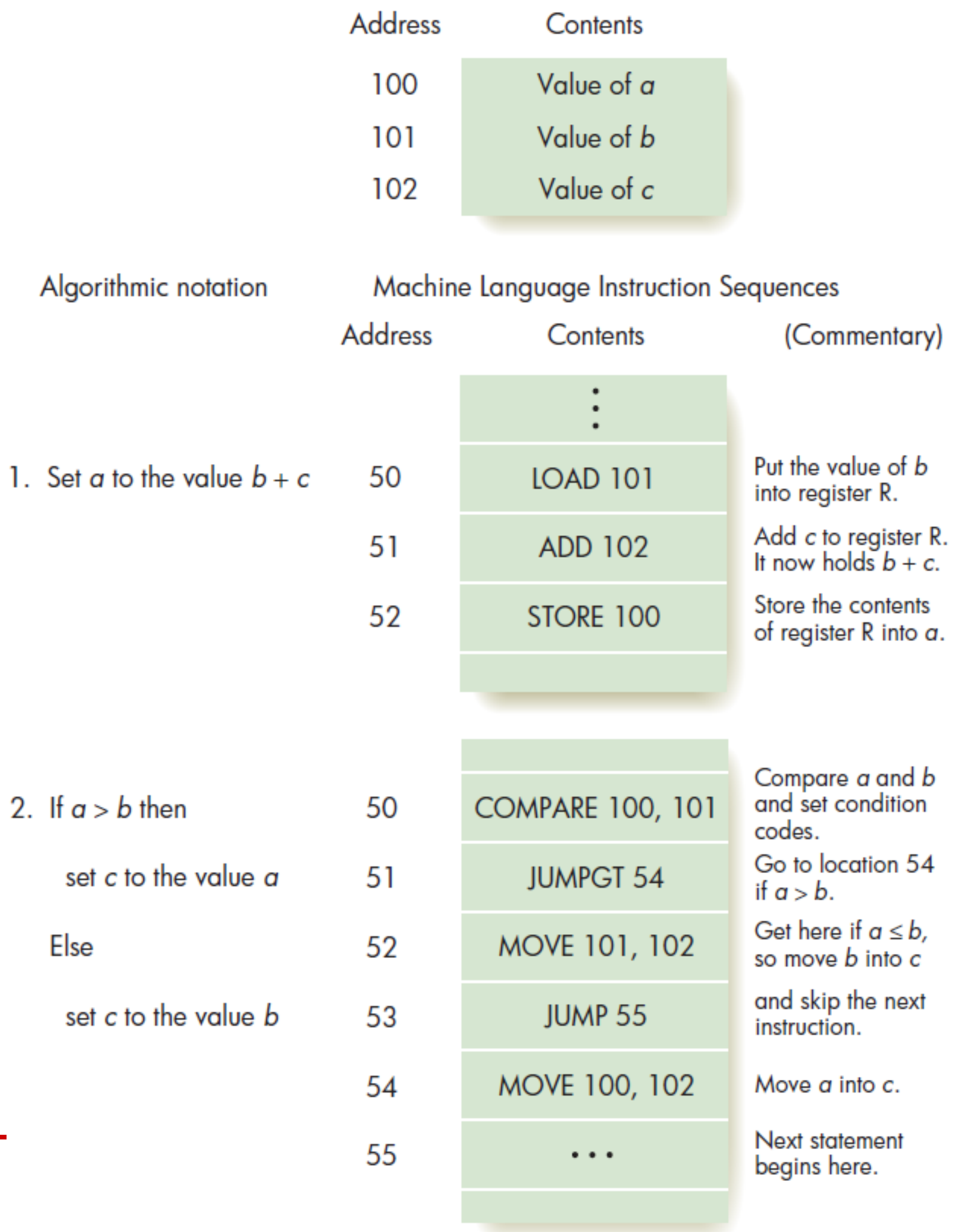
More on Instructions



- LOAD X -> Load register R with the contents of memory cell X
- STORE X -> Store register R into memory cell X
- MOVE X, Y -> Copy the contents of X into Y
- ADD X -> Add contents of X to the contents of R
- ADD X,Y -> Add contents of X to the contents of Y, and put the result in register R
- COMPARE X, Y-> Set GT (greater than), EQ (equal) and LT (less than) condition codes
- JUMP X-> Jump unconditionally to the instruction in cell X
- JUMPGT X-> Jump, if GT=1, to the instruction in cell X
- HALT

Machine Language Instructions (continued)

- Types of machine instructions:
  - Data transfer
    - Move values to and from memory and registers
  - Arithmetic/logic
    - Perform ALU operations that produce numeric values
  - Compares
    - Set bits of compare register to hold result
  - Branches
    - Jump to a new memory address to continue processing

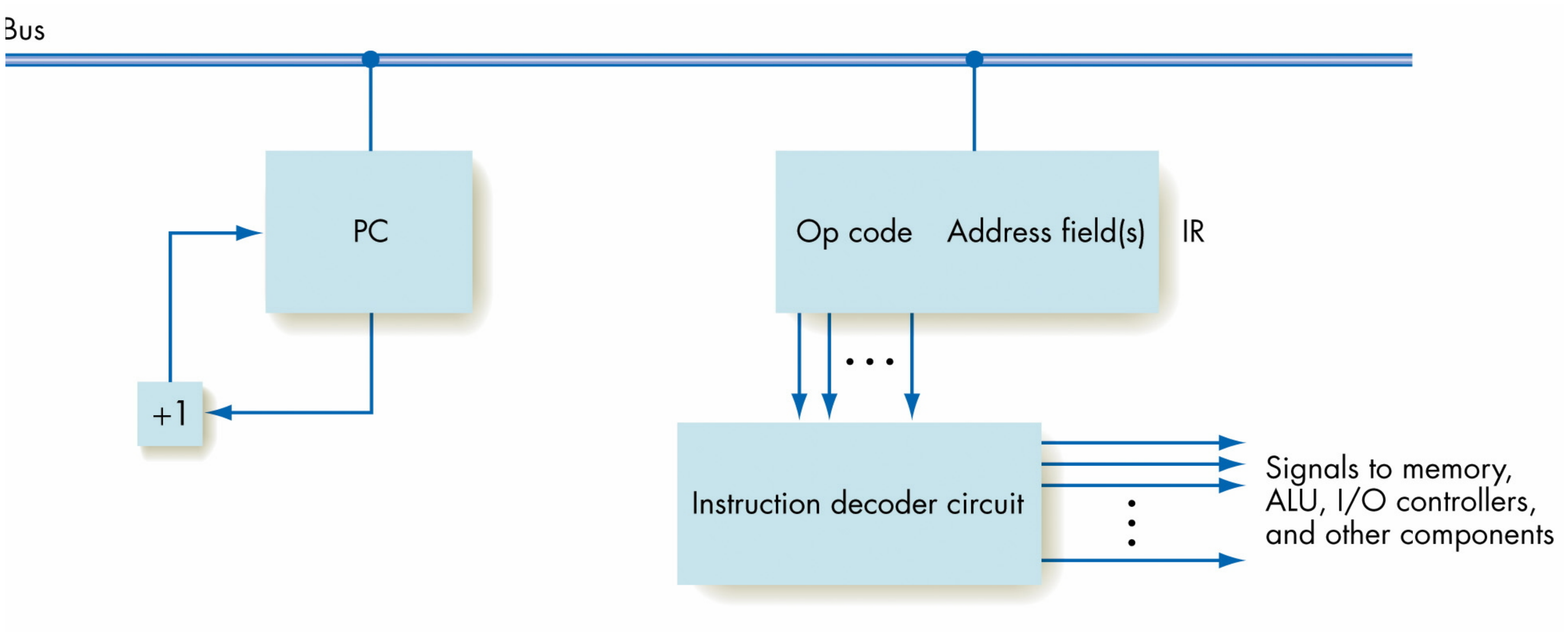


Address	Contents
100	Value of *a*
101	Value of *b*
102	Value of *c*

Algorithmic notation	Address	Contents	(Commentary)
		⋮	
1. Set *a* to the value *b* + *c*	50	LOAD 101	Put the value of *b* into register R.
	51	ADD 102	Add *c* to register R. It now holds *b* + *c*.
	52	STORE 100	Store the contents of register R into *a*.
2. If *a* > *b* then	50	COMPARE 100, 101	Compare *a* and *b* and set condition codes.
set *c* to the value *a*	51	JUMPGT 54	Go to location 54 if *a* > *b*.
Else	52	MOVE 101, 102	Get here if *a* ≤ *b*, so move *b* into *c*
set *c* to the value *b*	53	JUMP 55	and skip the next instruction.
	54	MOVE 100, 102	Move *a* into *c*.
	55	. . .	Next statement begins here.

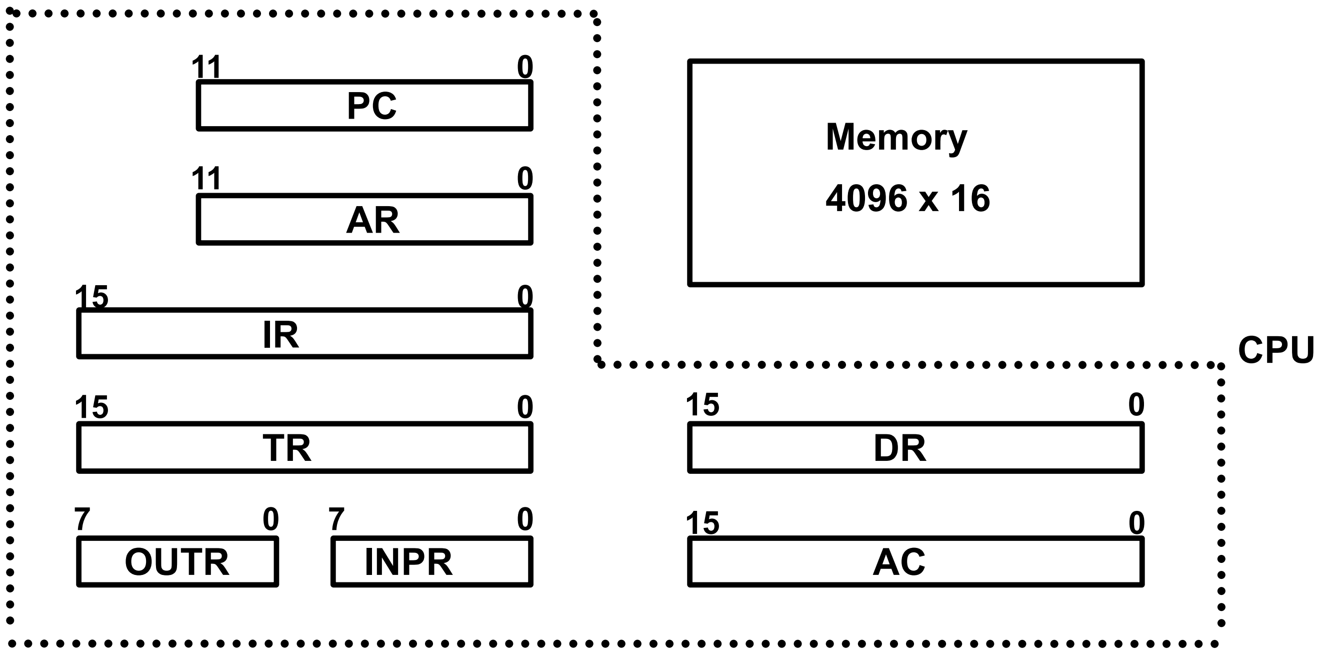
Control Unit Registers and Circuits

- Parts of control unit
  - Linked to other subsystems
  - Instruction decoder circuit (main responsibility)
  - Two special registers:
    - Program Counter (PC)
      - Stores the memory address of the next instruction to be executed
    - Instruction Register (IR)
      - Stores the code for the current instruction


Bus
PC
+1
Op code
Address field(s)
IR
. . .
Instruction decoder circuit
Signals to memory, ALU, I/O controllers, and other components

Organization of the Control Unit Registers and Circuits

Registers in a Basic Computer



List of BC Registers

DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

Main Types of Registers

- **User-accessible Registers**:
  - The most common division of user-accessible registers is into data registers and address registers.
- **Data registers**: hold numeric values such as integer and floating-point values.
  - In some older and low end CPUs, a special data register, known as the accumulator, is used implicitly for many operations.
- **Address registers:** hold addresses and are used by instructions that indirectly access memory.
- **Conditional registers:** hold truth values often used to determine whether some instruction should or should not be executed.
- **General purpose registers (GPRs)** can store both data and addresses, i.e., they are combined Data/Address registers.

http://en.wikipedia.org/wiki/Processor_register

Main Types of Registers (cont’d)

- **Floating point registers** (FPRs) store floating point numbers in many architectures.
- **Constant registers** hold read-only values such as zero, one, or pi.
- **Special purpose registers** (SPR) hold program state; they usually include the program counter (aka instruction pointer), stack pointer, and status register (aka processor status word). In embedded microprocessors, they can also correspond to specialized hardware elements.
- **Instruction registers** store the instruction currently being executed.

http://en.wikipedia.org/wiki/Processor_register

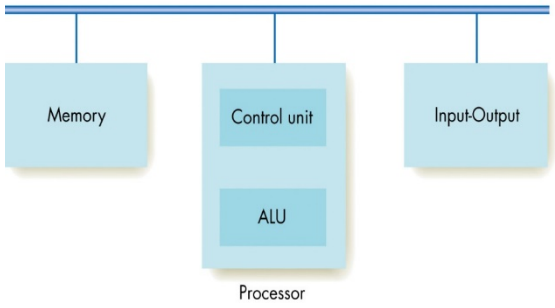
Some Architectures & The number of registers on them

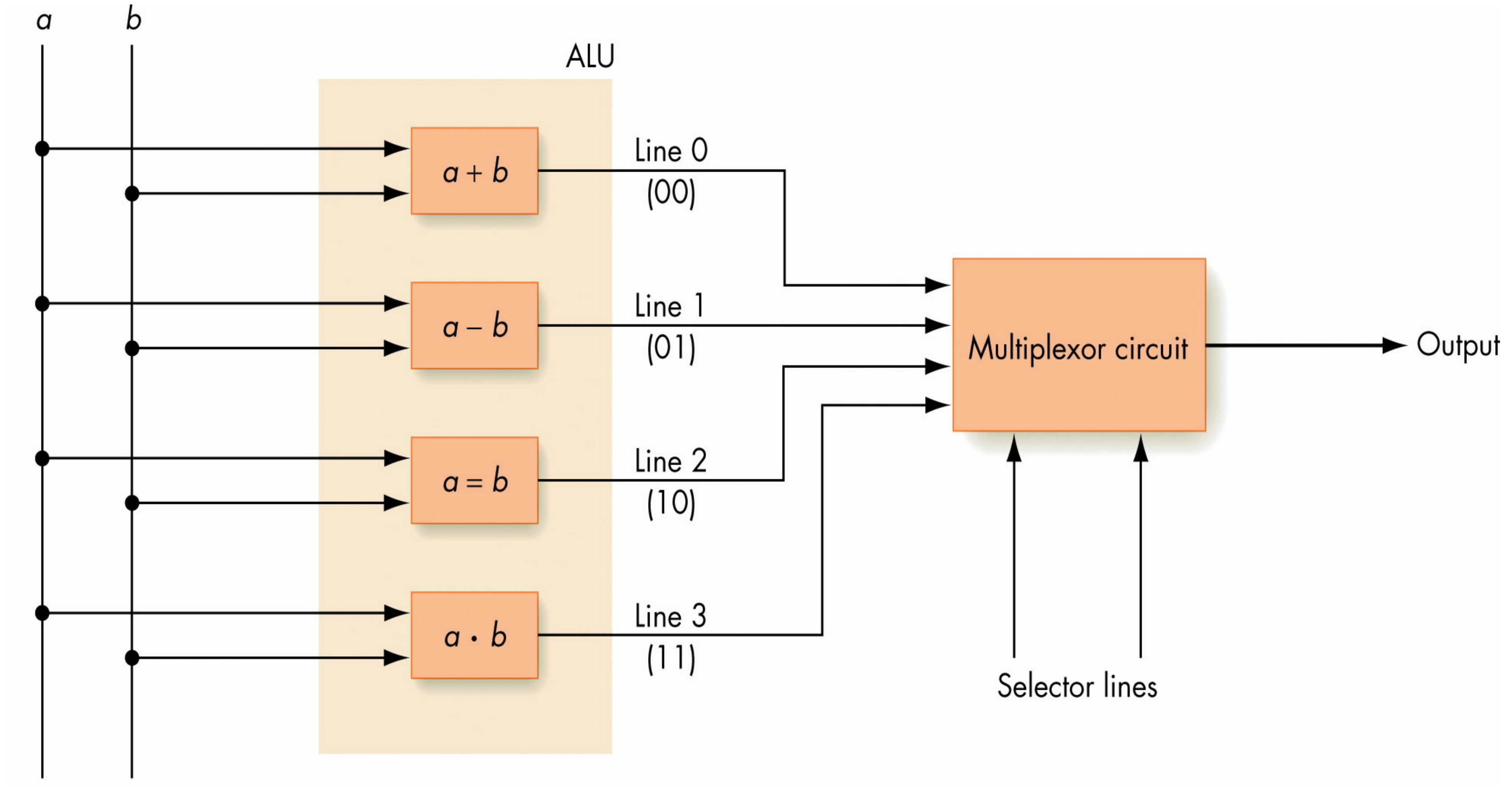
Architecture	Integer registers	FP registers
x86	8	8
x86-64	16	16
IBM/360	16	4
Z/Architecture	16	16
Itanium	128	128
UltraSPARC	32	32
IBM POWER	32	32
Alpha	32	32
6502	3	0
PIC microcontroller	1	0
AVR microcontroller	32	0
ARM	16	16

http://en.wikipedia.org/wiki/Processor_register

The Arithmetic/Logic Unit

- ***A**rithmetic and **L**ogic **U**nit*
  - *"Manufacturing" section of computer*
  - *Contains decision mechanisms and can make calculations+comparisons*
  - Actual computations are performed
- Primitive operation circuits
  - Arithmetic *[+, -, *, /]*
  - Comparison *[equality or CE, GT, LT, NEQ]*
  - Logic *[AND, OR, NOT, XOR]*
- Data inputs and results stored in registers
- Multiplexer selects desired output





Using a Multiplexor Circuit to Select the Proper ALU Result

(Not totally correct)

The Arithmetic/Logic Unit (continued)

- ALU process
  - Values for operations copied into ALU's input register locations
  - All circuits compute results for those inputs
  - Multiplexor selects the one desired result from all values (Not totally correct)
  - Result value copied to desired result register