# 1 Performance Equations and Power

## 1.1 Performance

- Performance = $\frac{1}{\text{Execution Time}}$
- Speedup, B over A = $\frac{\text{Performance}_B}{\text{Performance}_A}$
- Performance Improvement = Speedup − 1
- Execution Time = Cycle Time × Cycles
- Cycles = Instructions × CPI
- Clock Speed = $\frac{1}{\text{Cycle Time}}$

## 1.2 Power

- Total Power = Dynamic Power + Leakage Power
- Dynamic Power $\alpha$ Activity × Capacitance × Voltage$^2$ × Frequency

**Allocate array of five 32-bit integers**

```
.data
            .align 2
arraybuf:   .space 20
```

# 2 Memory

## 2.1 Memory Organization

1. **Scratchpad**: holds a number of values for extremely fast access by the processor, made up of 32 **registers**
2. **Register**: element of a scratchpad, holds 32 bits - can either be a primitive value or an address in memory

## 2.2 Procedures

1. **Activation Record**: area on the stack allocated by a procedure, $fp points to the start of the activation record and $sp points to the end.
2. **Arguments**: $a0 to $a4 are populated with arguments Procedure A wants to call Procedure B with.
3. **Return**: when Procedure B returns, it needs to put the return value into $v0 or $v1 and hand control back to Procedure A
4. **Jump and Link**: `jal Bstart` jumps and hands control to procedure `BStart`
5. **Jump Register**: `jr $ra` jumps to the register `$ra`, which holds the return address.
6. **Scratchpad**:
   (a) Caller should save $ra, $a0..., $t0... $fp (if required).
   (b) Callee should save $s0...

(d) Write the line(s) of code that print this count. **(3 points)**
*Solution:*

```
li   $v0, 1
add  $a0, $t3, $zero
syscall
```

# 3 Numeric Representations

## 3.1 Unsigned Integers

1. **Decimal Numbers**: $3512 = 3 \times 10^3 + 5 \times 10^2 + 1 \times 10^1 + 2 \times 10^0$
2. **Binary Numbers**: $10101 = 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
3. **Hexadecimal**: $0x19af = 1 \times 16^3 + 9 \times 16^2 + 10 \times 16^1 + 15 \times 16^0$

## 3.2 Signed Integers

1. **Binary Numbers - Two's Complement**: first bit represents sign; the rest counts up from 0 (when positive) and up from $-2^{31}$ (when negative)

- 0000...0000 = 0
- 0111...1111 = $2^{31} - 1$
- 1000...0000 = $-2^{31}$
- 1111...1111 = $-1$
- $x_{31}x_{30}...x_0 = -2^{31}x_{31} + 2^{30}x_{30} + 2^{29}x_{29} + ... + 2^0x_0$
- Addition can be done just like decimal
- Let $\overline{x}$ be $x$ with bits inverted.
- $-x = \overline{x} + 1$

## 3.3 Floating Point Numbers

1. **Single-Precision Floating Point**:

   (a) Put the number into scientific notation in binary, of the form $\pm$1.fraction $\times$ $2^{\text{exponent}}$. Bits go from 31 to 0, left to right.

   (b) Bit 31 (S) stores the sign, 0 for positive and 1 for negative

   (c) Bits 30-23 (E) stores (exponent + bias) as an unsigned int, bias is 127

   (d) Bits 22-0 (F) stores the fraction section (right of the point) of the number, trailing 0s are added to the right as needed.

   (e) **Addition**: normalize the smaller exponent to match the larger exponent, add the fractions

2. **Double-Precision Floating Point**: same as single-precision, but 11 exponent bits, 52 fraction bits, bias is 1023.

# 4 Digital Design

## 4.1 Boolean Algebra

- **OR**: A + B $\iff$ A or B
- **AND**: A.B $\iff$ A and B
- **NOT**: $\overline{A}$ $\iff$ not A
- $\overline{A+B} = \overline{A}.\overline{B}$
- $\overline{A.B} = \overline{A} + \overline{B}$
- $A.(B+C) = (A.B) + (A.C)$
- $A + (B.C) = (A+B).(A+C)$
- **Sum of Products**: using a truth table, pick all true conditions and OR them together

## 4.2 Ripple-Carry Adder

1. A 1-bit adder takes in a carry (carry-in), puts out a carry (carry-out), and adds two digits.
2. Ripple-Carry connects the adders together; the carry-out of one adder serves as the carry-in of the next adder. This is similar to how we do addition, one pair of digits at a time and carrying when needed.

## 4.3 Carry-Lookahead Adder

1. Instead of waiting for the previous adder to put in a carry, we calculate whether there exists a carry ahead of time.
2. $c_{i+1} = a_i.b_i + (a_i.b_i).c_i$: $a_i$ is $i^{th}$ bit of a, $b_i$ is $i^{th}$ bit of b, $c_i$ is whether $i^{th}$ bit has a carry
3. Can chunk bits into 4 - do those 4 bits generate a carry?

# 5 Finite State Machines

1. **Finite State Machine**: the machine takes **input** and stores a **state**, using that information to move to a new state
2. **Finite State Table**: enumerate all possible permutations of current state and inputs, record the next state that the machine stores (output)
3. **Finite State Diagram**:

   (a) What are possible output states? Draw a bubble for each.

   (b) What are inputs? What values can those inputs take?

   (c) For each state, what do I do for each possible input value? Draw an arc out of every bubble for every input value to represent a transition to a different state.