# B38DF
# Computer Architecture and Embedded Systems

## Tutorial 1

**Alexander Belyaev**

Heriot-Watt University
School of Engineering & Physical Sciences
Electrical, Electronic and Computer Engineering

E-mail: a.belyaev@hw.ac.uk
Office: EM2.29

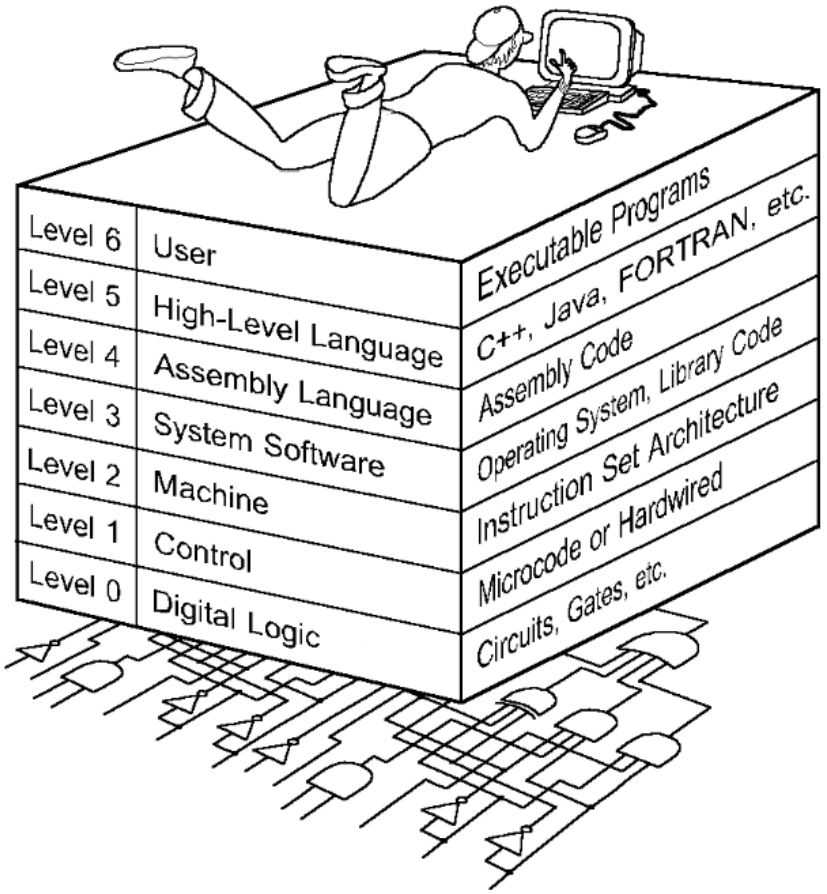
# Question 1

Explain each of the following terms in your own words:

a. Translator.

b. Interpreter.

c. Virtual machine.



# Answer to Q1

1. Explain each of the following terms in your own words:

   a. Translator: A translator converts programs in one language to another.

   b. Interpreter: An interpreter carries out a program instruction by instruction.

   c. Virtual machine: A virtual machine is a conceptual machine, one that does not exist.

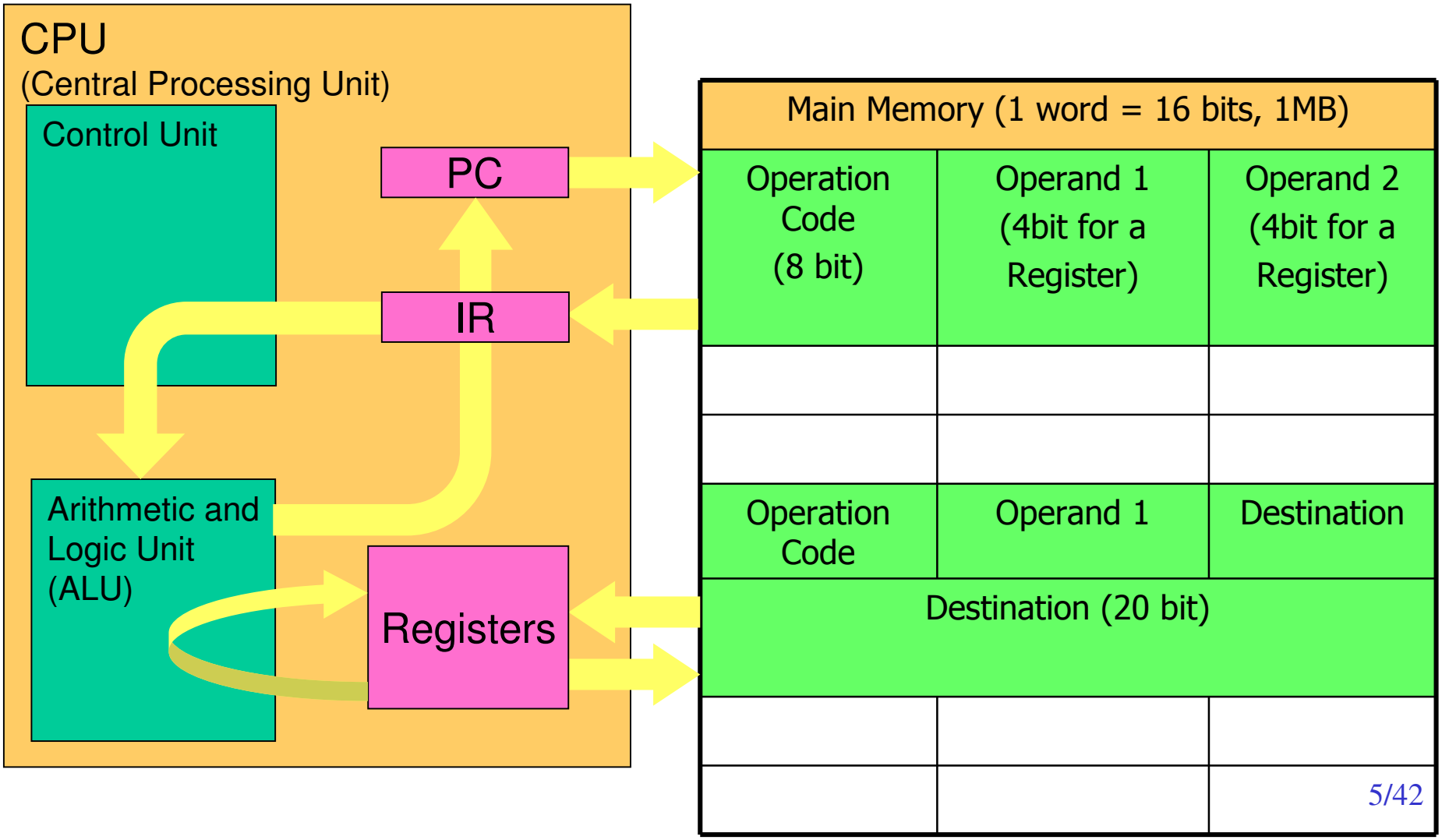
# Question 2

The CPU executes each instruction in a series of small steps. Describe briefly these steps.

# Question 2

The CPU executes each instruction in a series of small steps. Describe briefly these steps.

**Fetch**
**Decode**
**Execute**



# Answer to Q2

The CPU executes each instruction in a series of small steps. These steps are

1. Fetching the next instruction from memory into the instruction register.
2. Changing the program counter to point to the following instruction.
3. Determining the type of instruction just fetched.
4. If the instruction uses a word in memory, determining where it is.
5. Fetching the word, if needed, into a CPU register.
6. Executing the instruction.
7. Going to step 1 to begin executing the following instruction.

# Question 3

What is the purpose of step 2 in the list below? What would happen if this step were omitted?

1. Fetching the next instruction from memory into the instruction register.
2. Changing the program counter to point to the following instruction.
3. Determining the type of instruction just fetched.
4. If the instruction uses a word in memory, determining where it is.
5. Fetching the word, if needed, into a CPU register.
6. Executing the instruction.
7. Going to step 1 to begin executing the following instruction.

# Answer to Q3

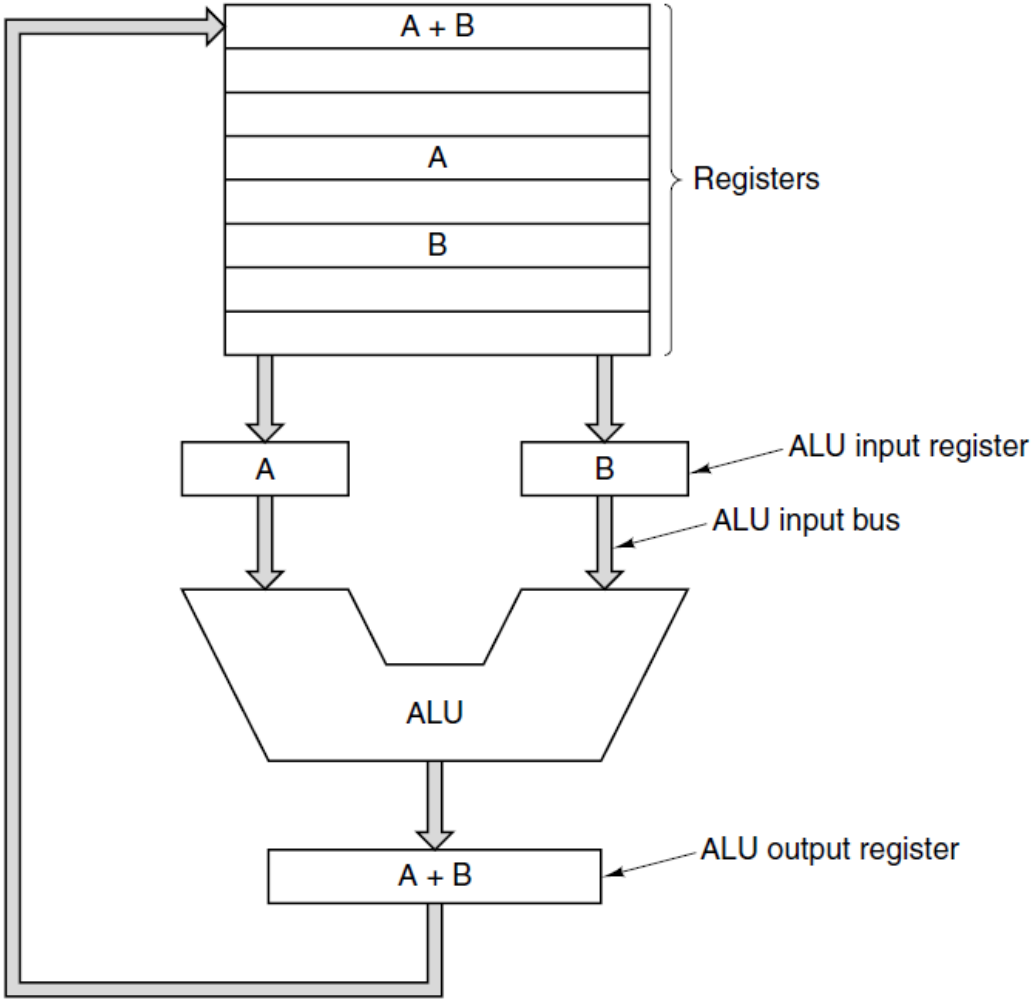
What is the purpose of step 2 in the list below? What would happen if this step were omitted?

1. Fetching the next instruction from memory into the instruction register.
2. Changing the program counter to point to the following instruction.
3. Determining the type of instruction just fetched.
4. If the instruction uses a word in memory, determining where it is.
5. Fetching the word, if needed, into a CPU register.
6. Executing the instruction.
7. Going to step 1 to begin executing the following instruction.

The program counter must be incremented to point to the next instruction. If this step were omitted, the computer would execute the initial instruction forever.

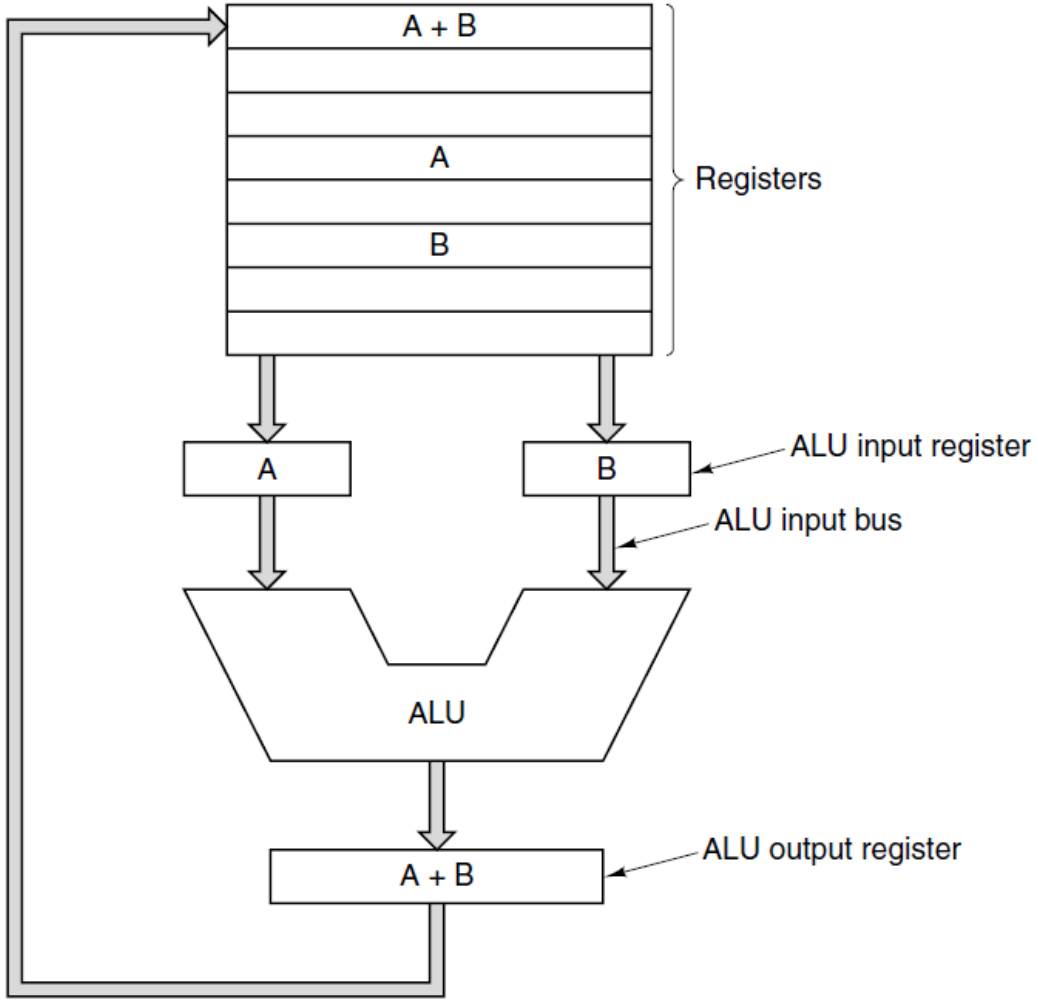
# Question 4

Consider the operation of a machine with the data path on the left. Suppose that loading the ALU input registers takes 5 nsec, running the ALU takes 10 nsec, and storing the result back in the register scratchpad takes 5 nsec. What is the maximum number of MIPS this machine is capable of in the absence of pipelining?



# Answer to Q4

Consider the operation of a machine with the data path on the left. Suppose that loading the ALU input registers takes 5 nsec, running the ALU takes 10 nsec, and storing the result back in the register scratchpad takes 5 nsec. What is the maximum number of MIPS this machine is capable of in the absence of pipelining?




The data path cycle is 20 nsec. The maximum number of data path cycles/sec is thus 50 million. The best the machine could do is thus 50 MIPS.

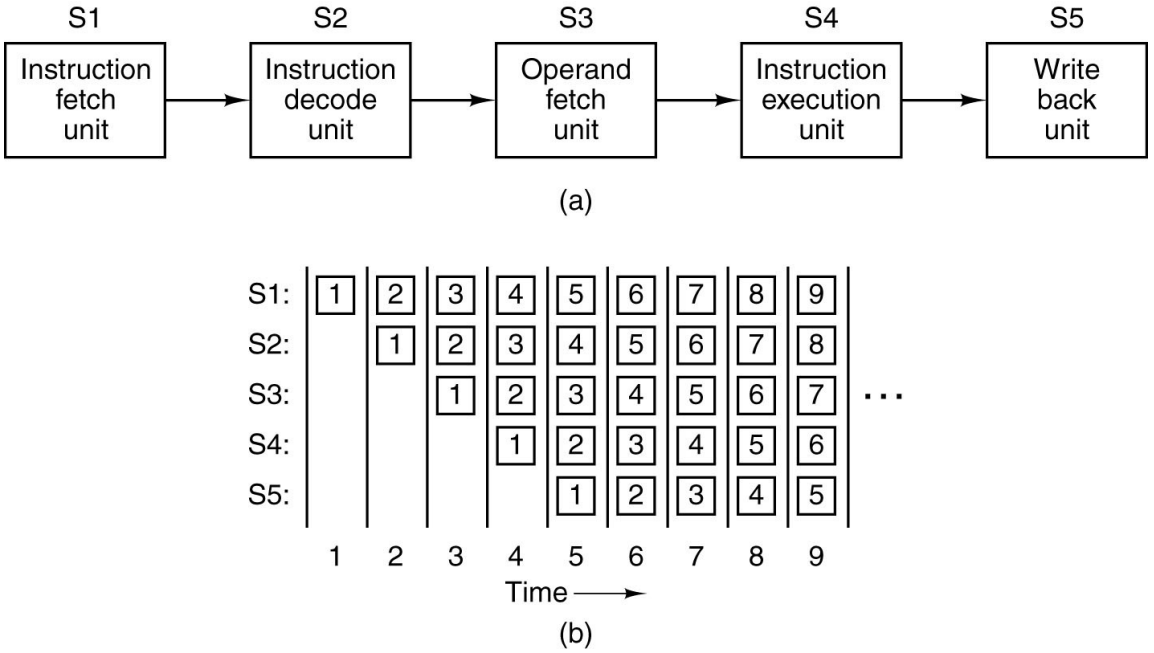
# Question 5

On computer 1, all instructions take 10 nsec to execute. On computer 2, they all take 5 nsec to execute. Can you say for certain that computer 2 is faster? Discuss.

# Answer to Q5

On computer 1, all instructions take 10 nsec to execute. On computer 2, they all take 5 nsec to execute. Can you say for certain that computer 2 is faster? Discuss.

You cannot say anything for sure. If computer 1 has a five-stage pipeline, it can issue up to 500 million instructions/second. If computer 2 is not pipelined, it cannot do any better than 200 million instructions/sec. Thus without more information, you cannot say which is faster.



# Question 6

One of the consequences of von Neumann's idea to store the program in memory is that programs can be modified, just like data. Can you think of an example where this facility might have been useful? (*Hint*: Think about doing arithmetic on arrays.)

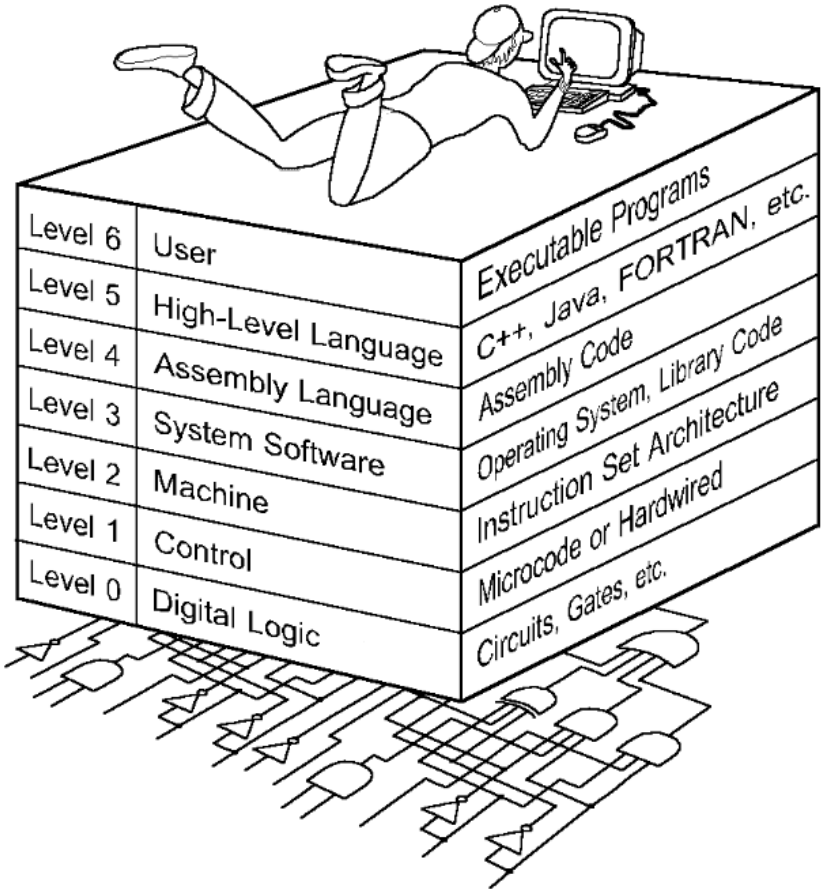
# Answer to Q6

One of the consequences of von Neumann's idea to store the program in memory is that programs can be modified, just like data. Can you think of an example where this facility might have been useful? (*Hint*: Think about doing arithmetic on arrays.)

A typical example is a program that computes the inner product of two arrays, *A* and *B*. The first two instructions might fetch *A*[0] and *B*[0], respectively. At the end of the iteration, these instructions could be incremented to point to *A*[1] and *B*[1], respectively. Before indexing and indirect addressing were invented, this was done.

# Question 7

Consider a multilevel computer in which all the levels are different. Each level has instructions that are *m* times as powerful as those of the level below it; that is, one level:*r* instruction can do the work of *m* level:(*r* – 1) instructions. If a level:1 program requires *k* seconds to run, how long would equivalent programs take at levels 2, 3, and 4, assuming *n* level:*r* instructions are required to interpret a single level:(*r* + 1) instruction?



# Answer to Q7

Consider a multilevel computer in which all the levels are different. Each level has instructions that are $m$ times as powerful as those of the level below it; that is, one level:$r$ instruction can do the work of $m$ level:$(r-1)$ instructions. If a level:1 program requires $k$ seconds to run, how long would equivalent programs take at levels 2, 3, and 4, assuming $n$ level:$r$ instructions are required to interpret a single level:$(r+1)$ instruction?

Each level of interpretation slows down the machine by a factor of $n/m$. Thus, the execution times for levels 2, 3, and 4 are $kn/m$, $kn^2/m^2$, and $kn^3/m^3$, respectively.

# Question 8

Consider a computer with identical interpreters at levels 1, 2, and 3. It takes an interpreter *n* instructions to fetch, examine, and execute one instruction. A level:1 instruction takes *k* nanoseconds to execute. How long does it take for an instruction at levels 2, 3, and 4?

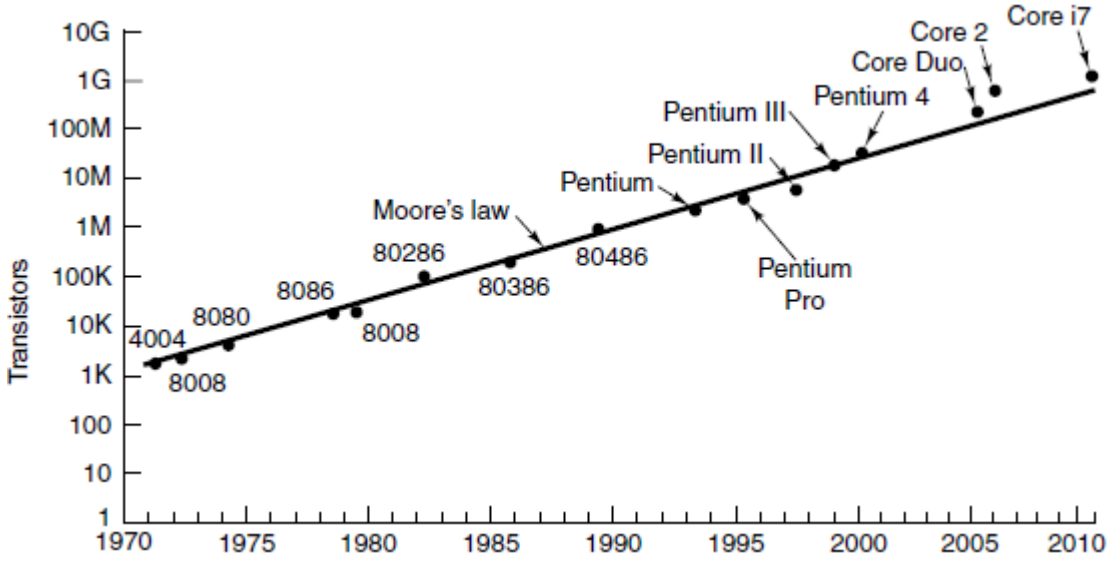
# Answer to Q8

Consider a computer with identical interpreters at levels 1, 2, and 3. It takes an interpreter $n$ instructions to fetch, examine, and execute one instruction. A level:1 instruction takes $k$ nanoseconds to execute. How long does it take for an instruction at levels 2, 3, and 4?

You lose a factor of $n$ at each level, so instruction execution times at levels 2, 3, and 4 are $kn$, $kn^2$, and $kn^3$, respectively.

# Question 9

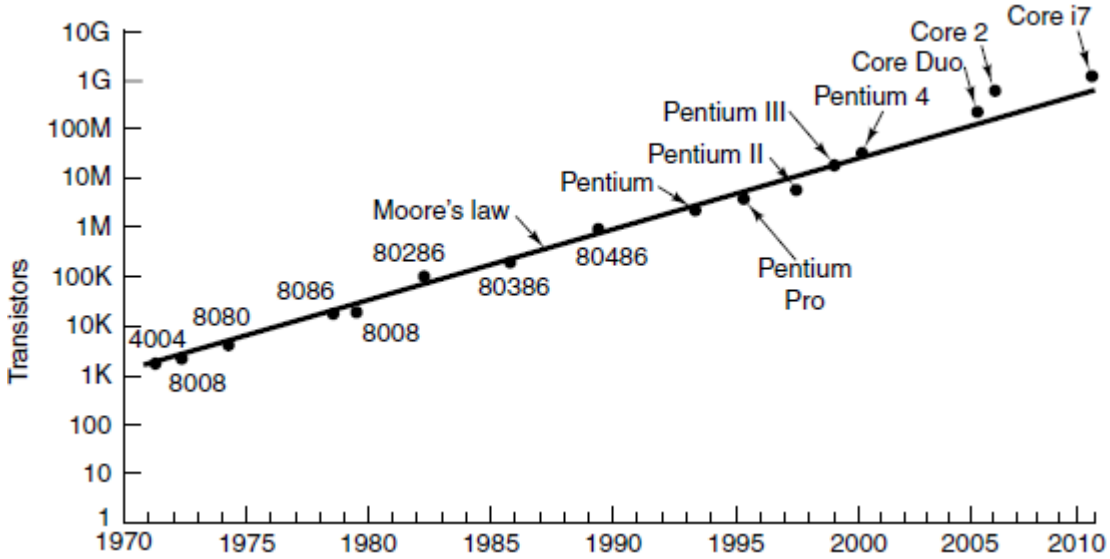
Today, Moore's law is often expressed as the doubling of the number of transistors every 18 months.




At a certain point in time, a transistor on a chip was 0.1 micron (1 μm=$10^{-6}$ metre) in diameter. According to Moore's law, how big would a transistor be on next year's model?

# Answer to Q9

Today, Moore's law is often expressed as the doubling of the number of transistors every 18 months.




At a certain point in time, a transistor on a chip was 0.1 micron (1 μm=$10^{-6}$ metre) in diameter. According to Moore's law, how big would a transistor be on next year's model?

According to Moore's law, next year the same chip will have 1.6 times as many transistors. This means that the area of each transistor will be 1/1.6 or 0.625 times the size of this year's transistors. Since the area goes like the square of the diameter, the diameter of next year's transistors must be 0.079 microns.

$$\sqrt{0.625} \approx 0.79$$

# Question 10

Developments in the computer industry are often cyclic. Originally, computing was centralized on large mainframe computers. Can you demonstrate a cyclic behaviour here?

# Answer to Q10

Developments in the computer industry are often cyclic. Originally, computing was centralized on large mainframe computers. Can you demonstrate a cyclic behaviour here?

After mainframes, we had personal computers. A current development is moving computation to the cloud, which is basically centralized computing all over again.

# Question 11

Consider three different processors P1, P2, and P3 executing the same instruction set with the clock rates and CPIs (CPI stands for the average number of clock cycles each instruction takes to execute) given in the following table.

| Processor | Clock rate | CPI |
|---|---|---|
| P1 | 2 GHz | 1.5 |
| P2 | 1.5 GHz | 1.0 |
| P3 | 3 GHz | 2.5 |

Which processor has the highest performance?

# Answer to Q11

Consider three different processors P1, P2, and P3 executing the same instruction set with the clock rates and CPIs (CPI stands for the average number of clock cycles each instruction takes to execute) given in the following table.

| Processor | Clock rate | CPI |
|---|---|---|
| P1 | 2 GHz | 1.5 |
| P2 | 1.5 GHz | 1.0 |
| P3 | 3 GHz | 2.5 |

Which processor has the highest performance?

$$\text{performance of P1 (instructions/sec)} = 2 \times 10^9 / 1.5 = 1.33 \times 10^9$$
$$\text{performance of P2 (instructions/sec)} = 1.5 \times 10^9 / 1.0 = 1.5 \times 10^9$$
$$\text{performance of P3 (instructions/sec)} = 3 \times 10^9 / 2.5 = 1.2 \times 10^9$$

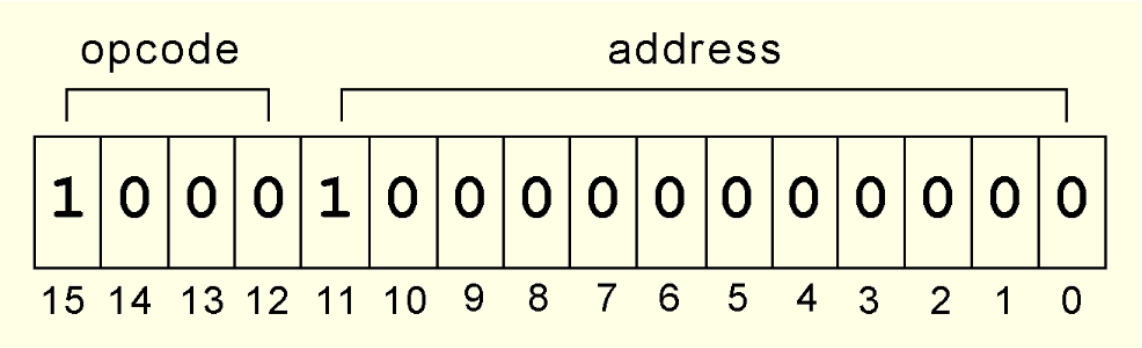
# Question 12

A digital computer has a memory unit with 24 bits per word. The instruction set consists of 150 different operations. All instructions have an operation code part (opcode) and an address part (allowing for only one address). Each instruction is stored in one word of memory.

a. How many bits are needed for the opcode?

b. How many bits are left for the address part of the instruction?

c. What is the maximum allowable size for memory?



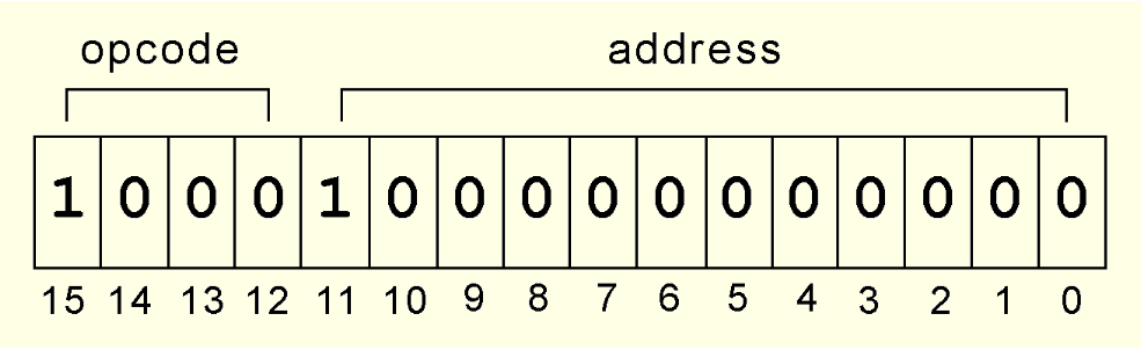
# Answer to Q12

A digital computer has a memory unit with 24 bits per word. The instruction set consists of 150 different operations. All instructions have an operation code part (opcode) and an address part (allowing for only one address). Each instruction is stored in one word of memory.

a. How many bits are needed for the opcode? 8, $2^7 = 128 < \mathbf{150} < 256 = 2^8$

b. How many bits are left for the address part of the instruction? 16=24-8

c. What is the maximum allowable size for memory? $2^{16}$



# Question 13

Assume a $2^{20}$ byte memory:

**a.** What are the lowest and highest addresses if memory is byte-addressable?

**b.** What are the lowest and highest addresses if memory is word-addressable, assuming a16-bit word?

**c.** What are the lowest and highest addresses if memory is word-addressable, assuming a 32-bit word?

# Answer to Q13

Assume a $2^{20}$ byte memory:

**a.** What are the lowest and highest addresses if memory is byte-addressable?

**b.** What are the lowest and highest addresses if memory is word-addressable, assuming a16-bit word?

**c.** What are the lowest and highest addresses if memory is word-addressable, assuming a 32-bit word?

a. There are $2^{20}$ bytes, which can all be addressed using addresses 0 through $2^{20}$-1 with 20 bit addresses.

b. There are only $2^{19}$ words and addressing each requires using addresses 0 through $2^{19}$ –1.

c. There are only $2^{18}$ words and addressing each requires using addresses 0 through $2^{18}$ –1.

# Question 14 **TRUE or FALSE ?**

a. A branch instruction changes the flow of information by changing the PC.

b. Registers are storage locations within the CPU itself.

c. An assembler is a program that accepts a symbolic language program and produces the binary machine language equivalent, resulting in a 1-to-1 correspondence between the assembly language source program and the machine language object program.

d. Program Counter is a special type of register in CPU which contains the code for the current instruction.

e. C++ and Java are examples for Assembly Languages.

# Answer to Q14 **TRUE or FALSE ?**

a. A branch instruction changes the flow of information by changing the PC. **T**

b. Registers are storage locations within the CPU itself. **T**

c. As assembler is a program that accepts a symbolic language program and produces the binary machine language equivalent, resulting in a 1-to-1 correspondence between the assembly language source program and the machine language object program. **T**

d. Program Counter is a special type of register in CPU which contains the code for the current instruction. **F**

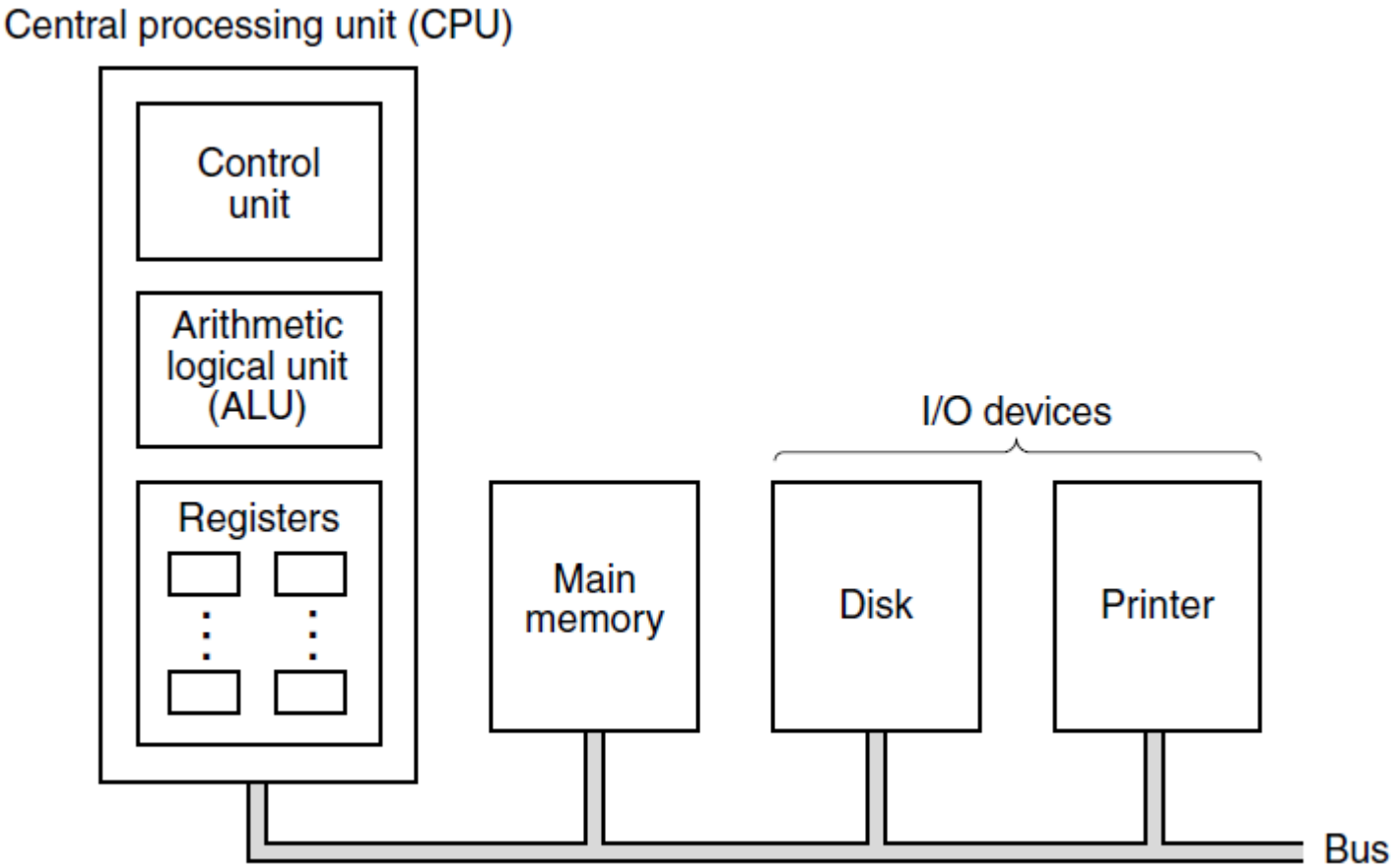
e. C++ and Java are examples for Assembly Languages. **F**

# Question 15

Draw a schematic diagram of computer (with a standard von Neumann architecture) showing the principal elements and the communication links between them.

# Answer to Q15

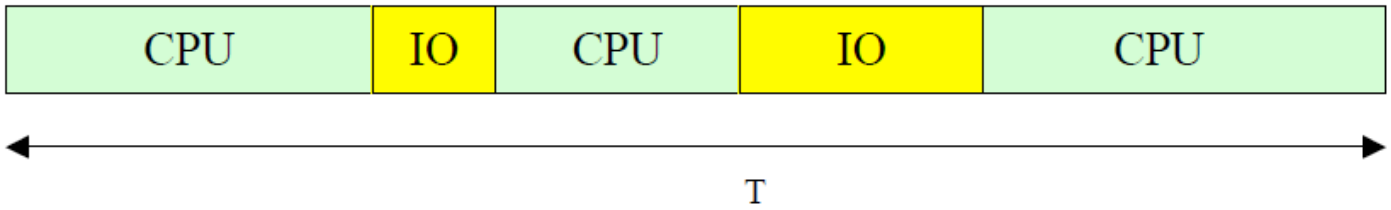
Draw a schematic diagram of computer (with a standard von Neumann architecture) showing the principal elements and the communication links between them.



# Question 16

Program execution time is made up of 75% CPU time and 25% I/O time. Which is the better enhancement: (a) Increasing the CPU speed by 50% or (b) reducing I/O time by half?

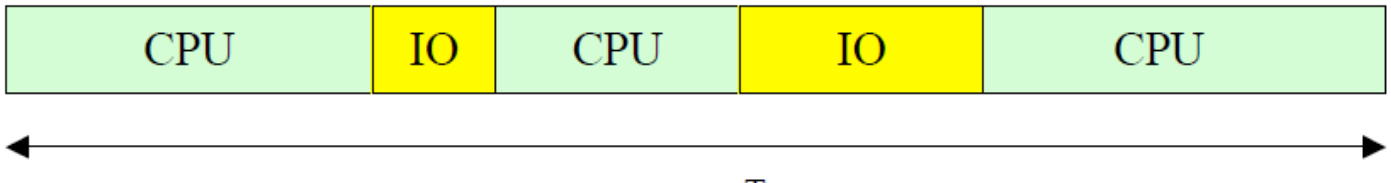
We assume that there are no overlaps between CPU and I/O operations



# Answer to Q16

Program execution time is made up of 75% CPU time and 25% I/O time. Which is the better enhancement: (a) Increasing the CPU speed by 50% or (b) reducing I/O time by half?

We assume that there are no overlaps between CPU and I/O operations




Increasing the CPU speed by 50%

$$T = T_{cpu} + T_{i/o} = 0.75T + 0.25T$$

$$T_{new} = T_{cpu}/1.5 + T_{i/o} = 0.5T + 0.25T = 0.75T$$

$$\text{Speedup} = T_{old}/T_{new} = 1.33$$

Halving the I/O Time

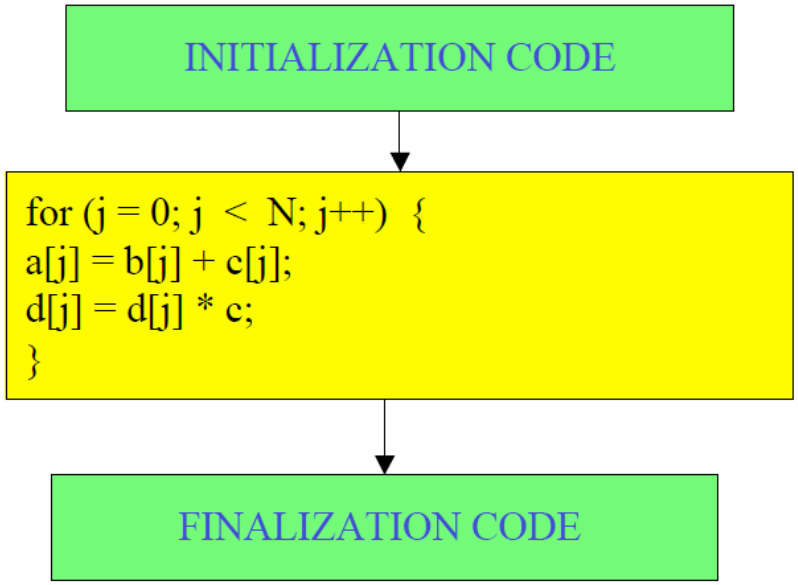
$$T = T_{cpu} + T_{i/o} = 0.75T + 0.25T$$

$$T_{new} = T_{cpu} + T_{i/o}/2 + = 0.75T + 0.125T = 0.825T$$

$$\text{Speedup} = T_{old}/T_{new} = 1.21$$

# Question 17

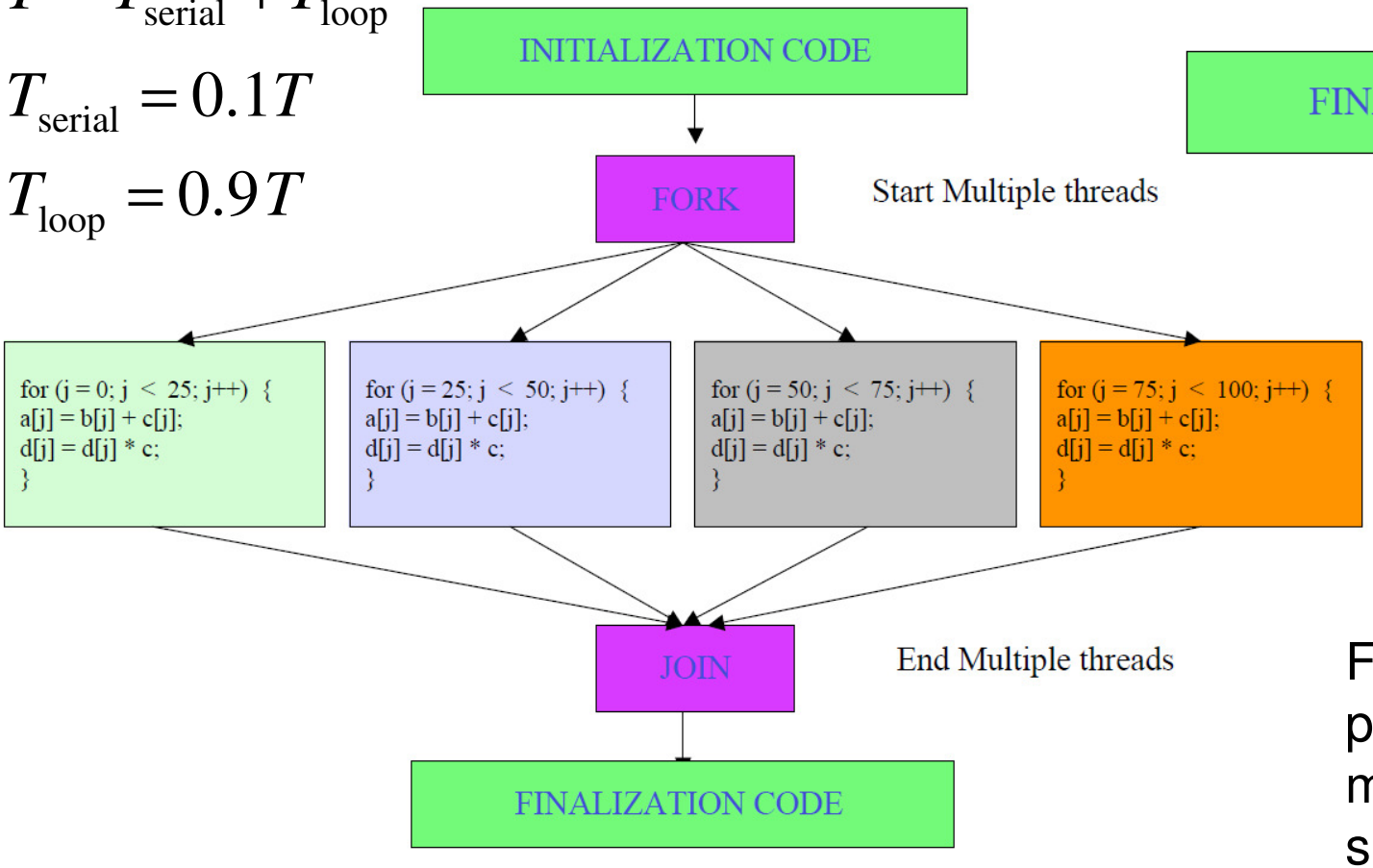
A program made up of 10% serial initialisation and finalization code. The remainder is a fully parallelizable loop of N iterations.




$$T = T_{\text{serial}} + T_{\text{loop}}$$

$$T_{\text{serial}} = 0.1T$$

$$T_{\text{loop}} = 0.9T$$




$$T_p = T_{\text{serial}} + T_{\text{loop}} / p$$

(System Calls for FORK/JOIN incur zero overhead)

Find speedups for p=2,4,8,16 and the maximal possible speedup.

# Answer to Q17

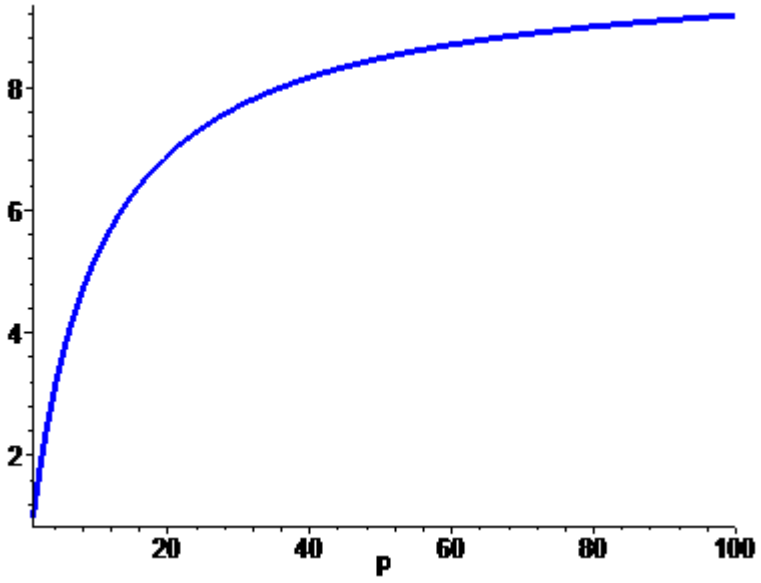
$p = 2$: $T_p = 0.1T + 0.9T/p = 0.55\ T$ Speedup = $T/0.55T = 1.8$

$p = 4$: $T_p = 0.1T + 0.9T/p = 0.325\ T$ Speedup = $T/0.325T = 3.0$

$p = 8$: $T_p = 0.1T + 0.9T/p = 0.2125\ T$ Speedup = $T/0.2125T = 4.7$

$p = 16$: $T_p = 0.1T + 0.9T/p = 0.15625\ T$ Speedup = $T/0.15625T = 6.4$

Maximal speedup is equal 10 and it is approaching when the number of processors goes to infinity.



# Question 18

Given a byte, 11001101, add even parity bit.

# Answer to Q18

Given a byte, 11001101, add even parity bit.

11001101 **1** Even parity bit is 1 since the number of 1's in the given byte is odd.

# Question 19

Assume we wish to create a code using 3 information bits, 1 parity bit (appended to the end of the information), and even parity. List all legal code words in this code. What is the Hamming distance of your code?

# Question 20

Compute the Hamming distance of the following code:

0011010010111100
0000011110001111
0010010110101101
0001011010011110

# Question 21

Given four data bits 1011, generate the Hamming code word.

# Answer to Q21

Given four data bits 1011, generate the Hamming code word.

We need three extra bits: $P_1P_2 1 P_4 011$, where $P_1$, $P_2$ and $P_4$ are parity bits.

$P_1$ checks bits 1,3,5,7, so $P_1=0$

$P_2$ checks bits 2,3,6,7, so $P_2=1$

$P_4$ checks bits 4,5,6,7, so $P_4=0$

Thus the Hamming code word is **0110011**