Processor System Performance

- Let:
  - IF_i = Frequency of instruction I_i
  - CPI_i = Clock cycles required for I_i
  - τ = Clock cycle period in ns
  - MIPS = Average inst'ns per sec (10^6)

Execution Time and MIPS

- Execution time of an average instruction can be expressed as:

$$\sum_{\text{all } i} (IF_i \times CPI_i \times t), \text{ where } i \text{ is an instruction class}$$

- Thus:

$$MIPS = \frac{1}{\sum (IF_i \times CPI_i \times t)} \times 1000$$

Example:

A SISD with τ = 10ns and microcoded floating point instructions

- See Zargham Figure 1.9

Instruction Class	Instruction Frequency %(IF)	Cycles per Instruction (CPI)	Weighted CPI (IF * CPI)
Load and store	30.4	1.5	0.456
Integer add and substuct	10.0	1	0.1
Integer multiply and divide	3.8	10	0.38
Floating-point add and substruct	9.5	7	0.665
Floating-point multiply and divide	6.5	15	0.975
Logical	3.0	1	0.03
Branch	20.0	1.5	0.3
Compare, shift system	16.8	2	0.336

		Cycles per average instruction = 3.242	

Execution time of an average instruction:
3.242*τ = 3.242*10 nanoseconds = = 32.42 nanoseconds
MIPS = (1/ 32.42)*1000 = 30.845

High Performance

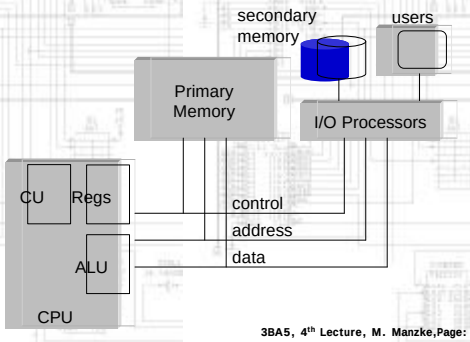
- High performance MISD or MIMD systems usually have pipelines for floating point arithmetic and are rated in millions of floating-point operations per second (MFLOPS)
- Example:
  - A vector processor with τ=5ns has separate floating point add and multiply pipes which can accept new input operands each clock period. What is the maximum sustainable MFLOPs?

Other Performance Measures

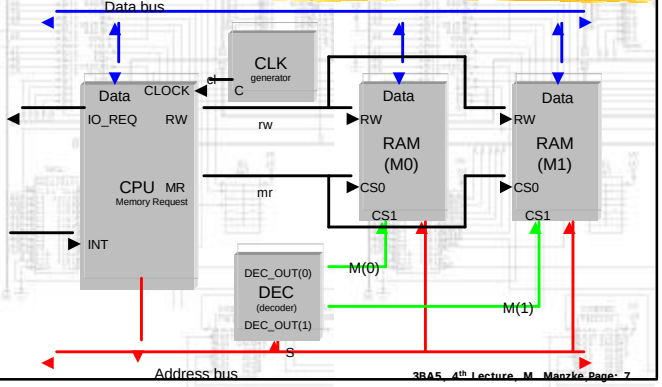
- Throughput = No of tasks completed per unit time
- Utilisation = (busy time / total time) * 100
- Response time = Time between a request and completion
- Memory bandwidth = Number of memory words accessed per second
- Memory access time = Time to access a word in memory in ns
- Memory size = Size of primary memory
- Latency = Time interval between a request and the start of a response

Organisation and Operation of a SISD

- Typical von Neumann Configuration



Simple System



Microprocessor

(see lecture four)

```
-- Michael Manzke
-- 18th October 2005
-- michael.manzke@cs.tcd.ie
-- 3ba5 tutorial
--
-- Example from:
-- Computer Architecture - single and parallel systems
-- Mehdi R. Zargham
-- Section 2.2 Design of a simpel microcomputer using VHDL
-- Figure 2.3 Structural representation of a simple microcomputer
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

architecture structure_view of microprocessor is
```

CPU and RAM Component

(see lecture four)

```
-- Mehdi R. Zargham, page 22
component CPU
  Port ( DATA : inout std_logic_vector(0 to 7);
         ADDR : out std_logic_vector(3 downto 0);
         CLOCK : in std_logic;
         INT : in std_logic;
         MR : out std_logic;
         RW : out std_logic;
         IO_REQ : out std_logic);
end component ;

component RAM
  Port ( DATA : inout std_logic_vector(0 to 7);
         ADDR : in std_logic_vector(2 downto 0);
         CS0 : in std_logic;
         CS1 : in std_logic;
         RW : in std_logic);
end component ;
```


Decoder and Clock Component

```
-- Mehdi R. Zargham, page 22
component DEC
  Port ( DEC_IN : in std_logic;
         DEC_OUT : out std_logic_vector(0 to 1));
end component ;

component CLK
  Port ( C : out std_logic);
end component ;
```


Port Map

(see lecture four)

```
-- Mehdi R. Zargham, page 22
signal M: std_logic_vector(0 to 1);
signal cl, mr, rw: std_logic;

begin

   PROCESSOR: CPU port map (DATA, ADDR, cl, INT, mr, rw, IO_REQ);
   M0: RAM port map (DATA, ADDR(2 downto 0), mr, M(0), rw);
   M1: RAM port map (DATA, ADDR(2 downto 0), mr, M(1), rw);
   DECODER: DEC port map (ADDR(3), M);
   CLOCK: port map (cl);

end structure_view;
```


RAM

(see lecture four)

```
-- Mehdi R. Zargham, page 22
architecture behavioral_view of RAM is
begin
  process
     type memory_unit is array(0 to7) of std_logic_vector(0 to 7);
     variable memory: memory_unit;
   begin
     while (CS0='1' and CS1='1') loop
        case RW is          -- RW = 0 means read operation
                            -- RW = 1 means write operation
          when '0' => DATA <= memory(intval(ADDR)) after 50 ns;
          when '1' => memory(intval(ADDR)) <= DATA after 60 ns;
        end case;
        wait on CS0, CS1, DATA, ADDR, RW;
     end loop
     wait on CS0, CS1;
  end process;
end behavioral_view;
```