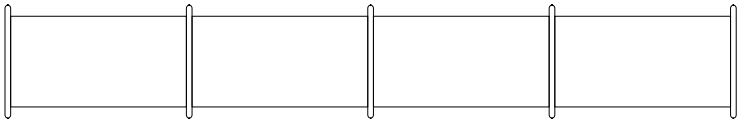
Pipelined Computers.



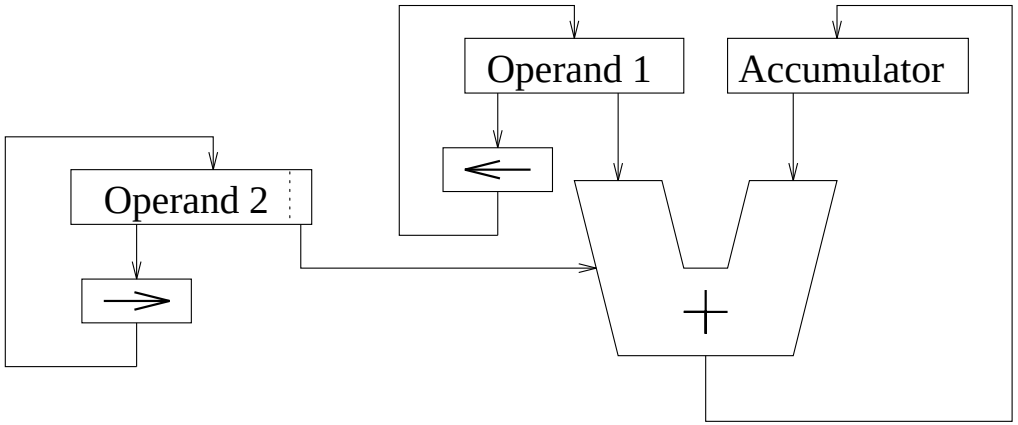
These machines can execute in parallel the various subfunctions of an operation in a manner similar to a factory assembly line.

Not all pipelined computers are considered as parallel computers. We shall restrict our definition to those that manipulate vector data structures. Such a computer is a Pipelined Vector Computer.

In order to understand a pipelined vector computer we will compare multiplication operations in sequential and vector parallel machines.

Multiplication in a Scalar Computer.

Consider the multiplication of two 16 bit integers to give a 32 bit integer. A simple 16 cycle algorithm is employed:

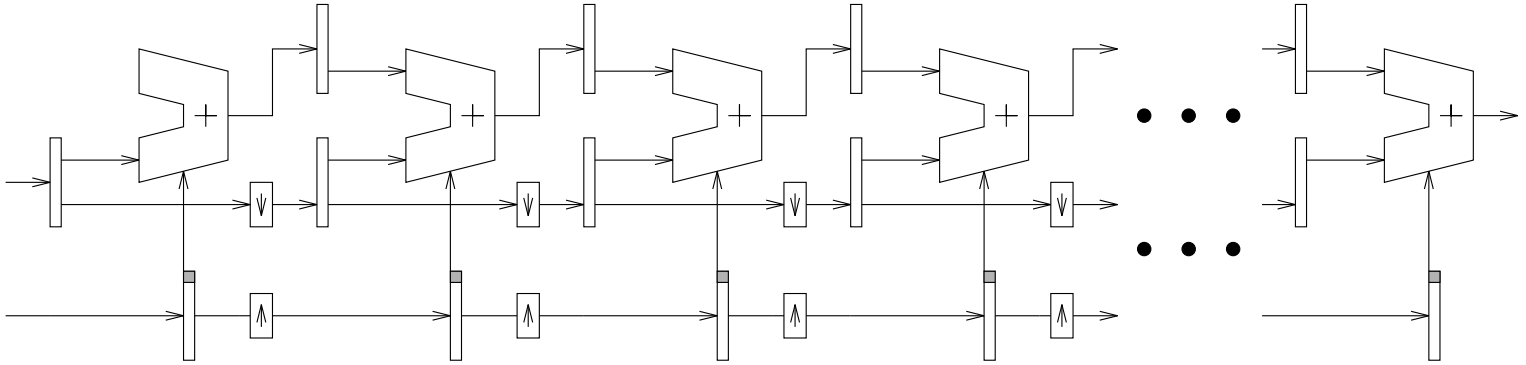


During each cycle

- Operand 1 is added to the Accumulator conditional on the least significant bit of Operand 2.
- Operand 1 is shifted right.
- Operand 2 is shifted left.

Pipeline Multiplier

In a pipelined architecture this process is unrolled:



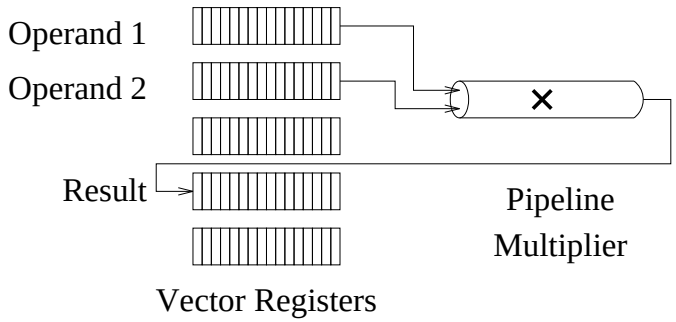
As in the scalar machine a single multiplication takes 16 cycles.

Where more than one multiplication is required a vector multiplication is used.

Multiplication in a Pipelined Vector Computer.

Vector Multiplication

The pipeline vector computer will multiply operands from vector registers giving a vector result.



- In each cycle a new pair of scalar operands will be presented to the pipeline.
- After an initial delay of 16 cycles results will be produced at a rate of one per cycle.
- The time to multiply vectors of n elements is $15 + n$ *cycles*[1]

[1] c.f. *16* x *n cycles* for scalar machine

Pipeline Granularity.

Our multiplier example performs the multiplication in 16 cycles.

- For a scalar machine, this involves a considerable saving in hardware over a multiplier which takes fewer cycles, due to the re-use of the same hardware.
- For the pipeline machine, the unrolled multiplier cannot re-use any hardware. Hence we can combine stages to produce an 8 or 4 cycle multiplier without any hardware cost.

A vector processor may have one or more pipelines, where each pipeline may be fixed (unifunctional) or re-configurable (multifunctional).

Across all pipelines, the maximum stage delay will be fixed, usually equal to one processor clock cycle.

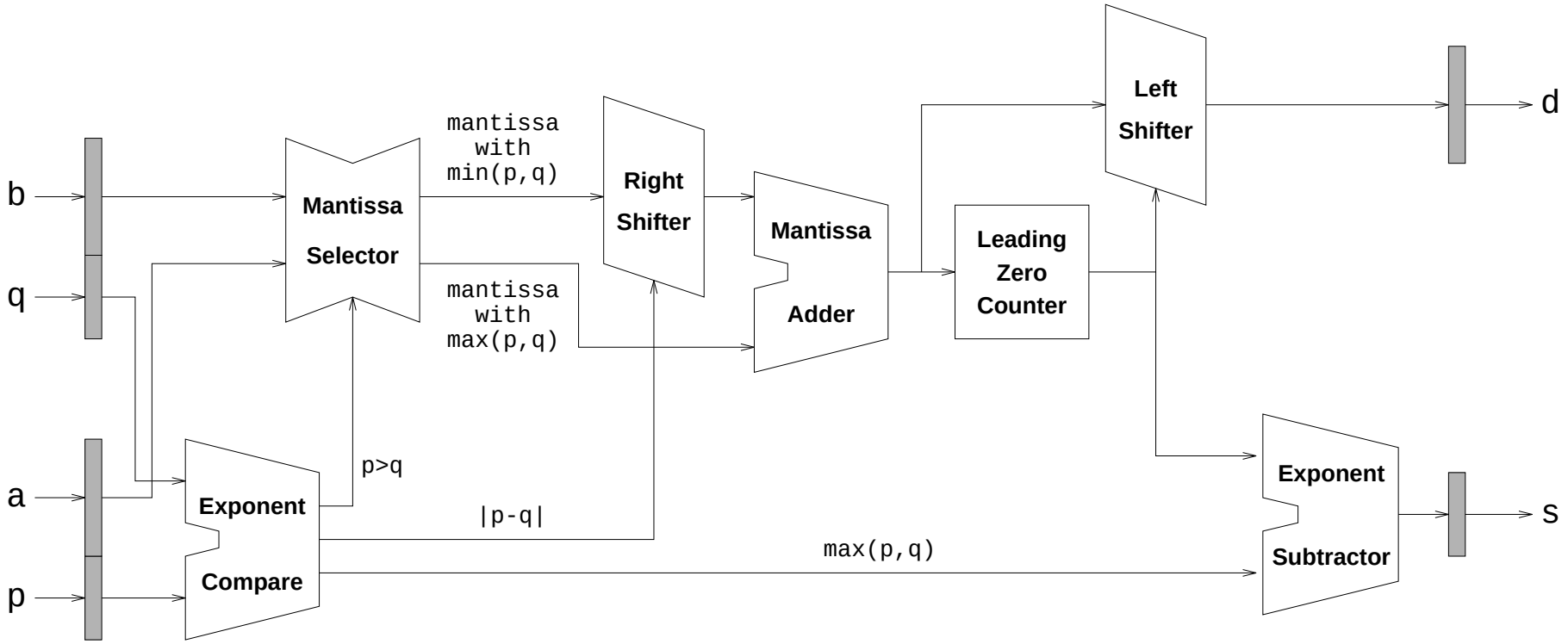
Thus the pipeline granularity will be chosen such that the individual stage delays are matched as closely as possible to this maximum.

Floating Point Addition Algorithm.

Consider the following addition:

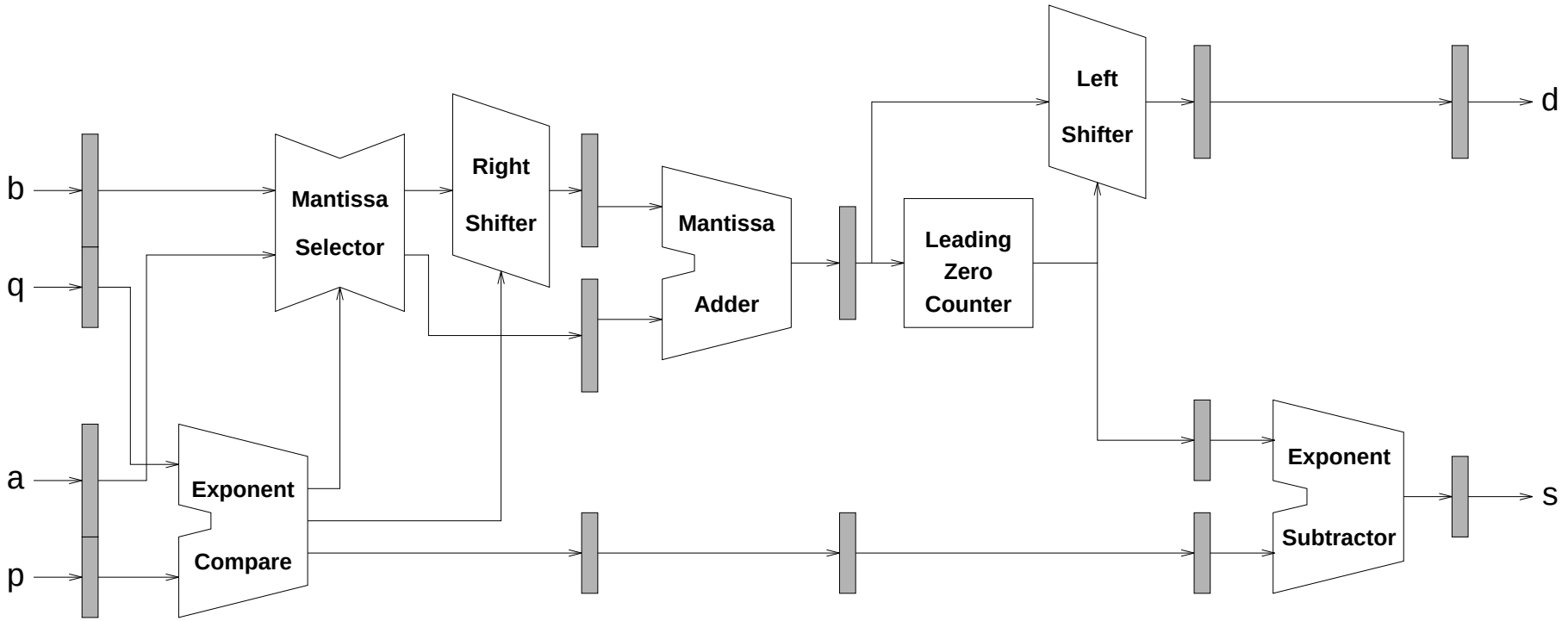
$$d \times 2^s = a \times 2^p + b \times 2^q$$

The standard algorithm involves a number of different sub-operations:



Pipeline for Floating Point Addition.

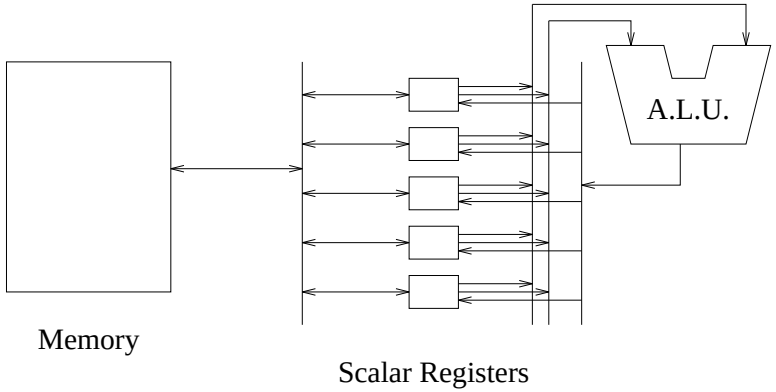
The following is a possible arrangement for a four stage pipeline:



CPU Power *vs* Memory Bandwidth

Scalar machine

Illustrated below is the architecture of a simple scalar machine with a register to register architecture. A well designed machine should balance memory bandwidth with C.P.U. power.

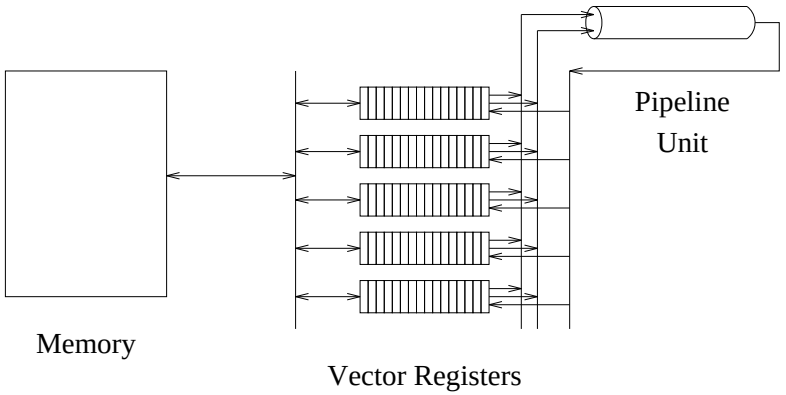


The memory may typically take three processor cycles for a single transfer while the A.L.U. may consume two values and produce one for every arithmetic operation.

Since the arithmetic operations take several cycles to produce a result this does not lead to any great imbalance.

CPU Power *vs* Memory Bandwidth

Vector Parallel machine

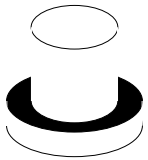


In the case of the pipeline processor we are aiming to calculate a new result every clock cycle.

Thus the pipeline unit expects three data transfers per cycle to and from the registers, while the memory can only support one data transfer every three cycles. The system is out of balance.

This explains why many vector parallel machines have very complex caching structures together with multiple banks of memory. These are the *speed at all costs* machines such as CRAY-1 and its successors.

Cray X-MP



Main processor unit (4 processor machine):

- Performance: 200 Mflops/s (approx.)
- Height: 2m
- Diameter: 1.5m
- Weight: 5 Mg

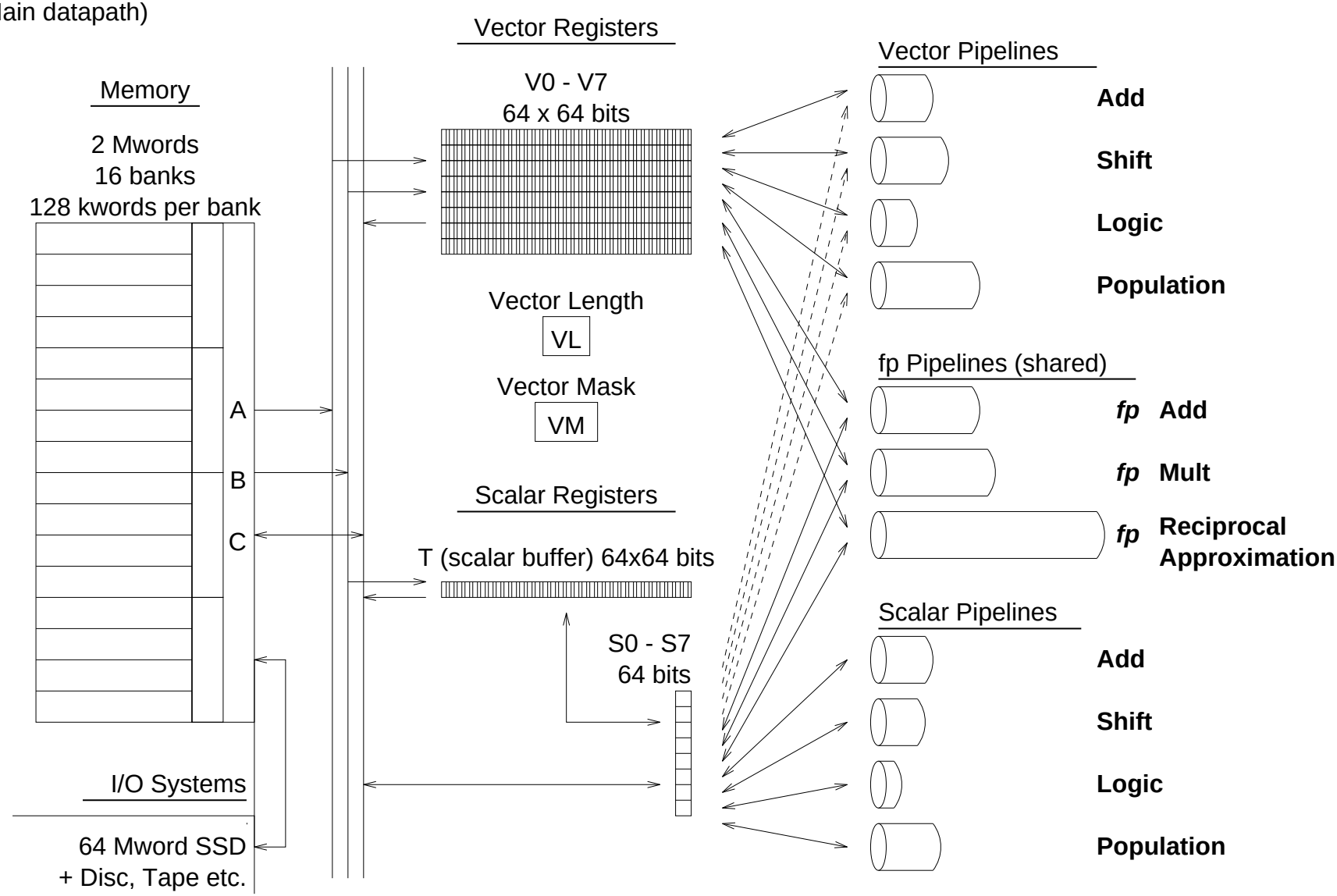
Support:

- 175 kVA generator
- Freon Cooling

We shall consider the single processor CRAY X-MP/12.

Cray X-MP/12

(Main datapath)



Register Structure

- Word length – 64 bits.
- V0 – V7
  8 off 64 word vector registers.
  Rapid update, pipeline to memory.
- S0 – S7
  8 off scalar registers.
- T0 – T63
  64 word scalar buffer.
  Rapid update, pipeline to memory.
  Allows fast feeding of scalar registers.

Pipeline Structure

- Vector Pipes
  - VL: Vector Length control
    A seven(?) bit register defining the length of the vectors to be processed.
    V5(0:31) = V1(0:31) * V3(0:31)
  - Vector & Scalar arguments
    Vector pipes can be fed with scalar and vector arguments to give vector results.
    V5(0:63) = V2(0:63) * S3
- Multiple pipes.
  Concurrent or chained operation.
- Accelerated pipes for scalar operands.
- Shared pipes for floating point operations.

Memory Structure

- 4 Port Memory
  - 3 for C.P.U.
  - 1 for I/O
- 16 Memory banks

  Reduces memory conflicts.
- Pipelined memory access

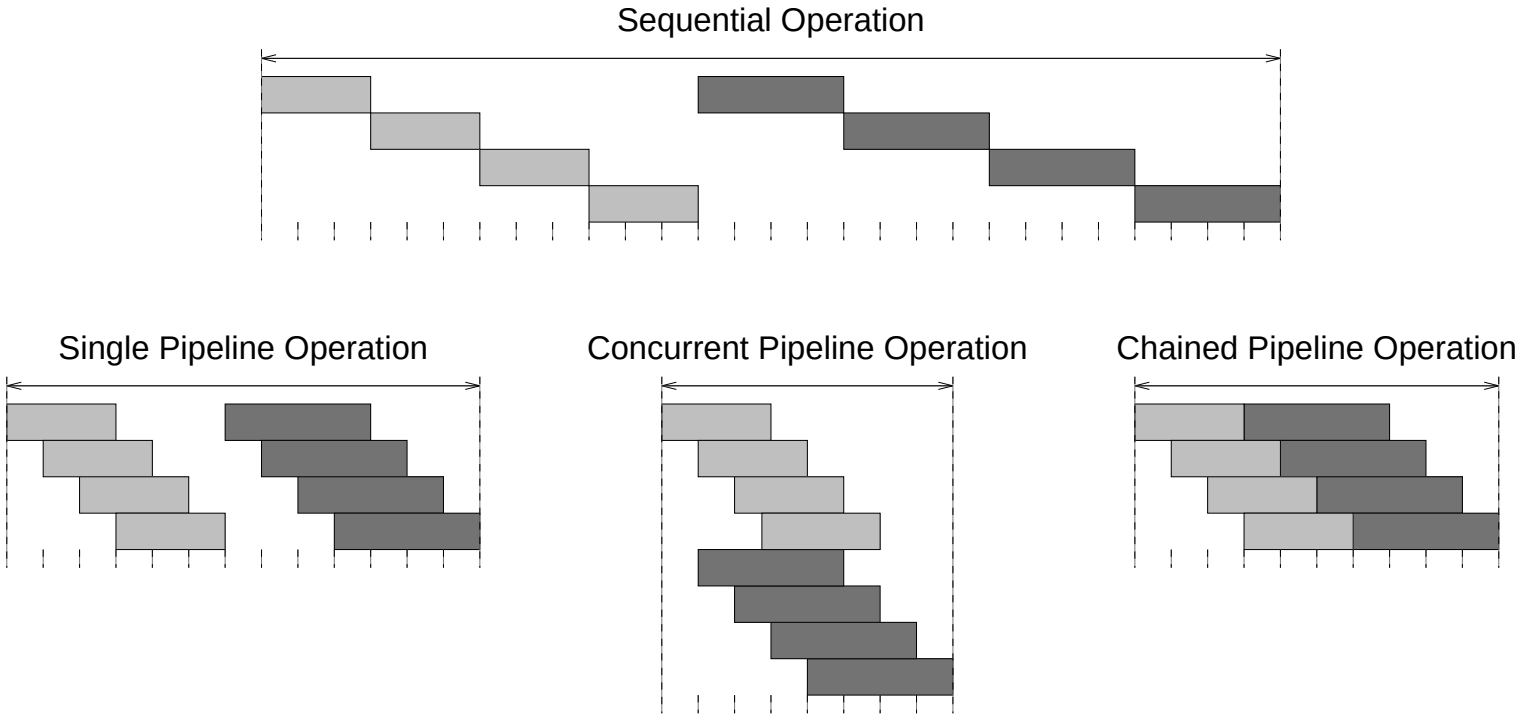
  A single memory access takes 4 cycles

  Pipelined access from four or more different banks gives one access per cycle per port.
- SSD Solid State Device

  Secondary Memory Device.

Concurrent & Chained Operations

Consider the execution of two vector operations.



- Sequential Operation. *no pipeline*

  No operation is started until the previous result has been obtained.

- Single Pipeline Operation. *single multifunctional pipeline*

  Two different vector operations are executed. Each pipeline operation yields one result per cycle. The first pipe is flushed before the second vector operation is started.

- Concurrent Pipeline Operation. *multiple pipeline – independent operations*

  Where different registers and different pipes are used, a new vector instruction may be issued every cycle.

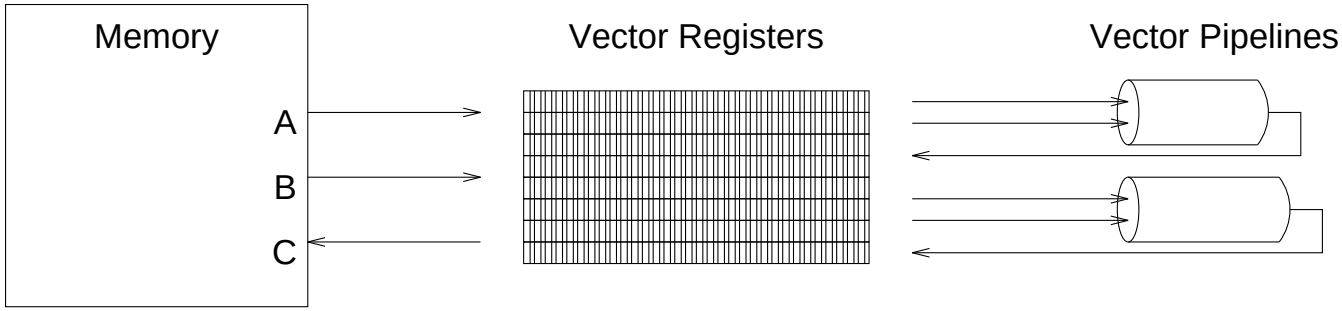
```
V0(0:3) = V1(0:3) + V2(0:3)
V3(0:3) = V4(0:3) * V5(0:3)
```

- Chained Pipeline Operation. *multiple pipeline – dependent operations*

  Here the second vector instruction uses the result of the first, the operations are chained together.

```
V0(0:3) = V1(0:3) + V2(0:3)
V3(0:3) = V0(0:3) * V5(0:3)
```

CPU Power *vs* Memory Bandwidth

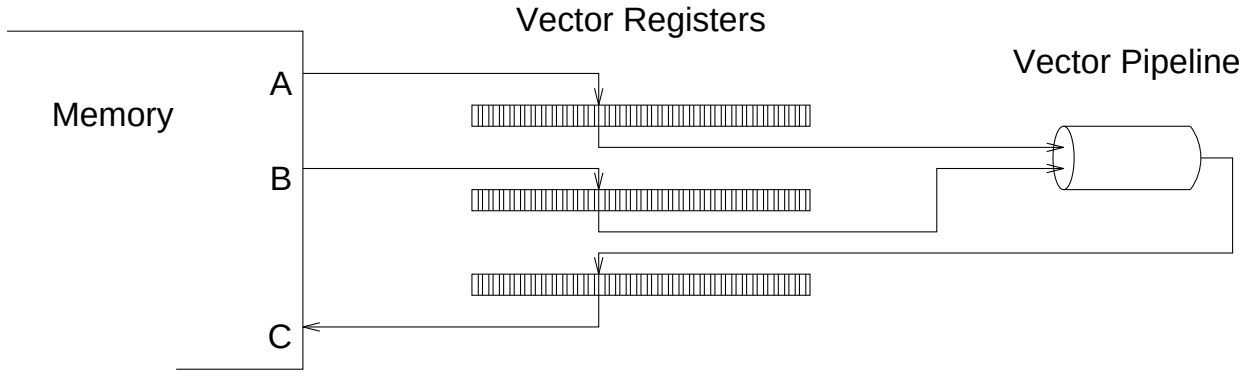


- CPU power.
  - Single active pipe – two operands and one result per cycle.
    Thus the pipeline/register transfer rate is three words per cycle.

- Memory Bandwidth
  - A, B & C memory ports can be used concurrently.
    Provided that there are no conflicts, the register/memory transfer rate is three words per cycle.

CPU Power *vs* Memory Bandwidth

Thus we can chain memory access and pipeline operations:[2]



Where more than one pipeline is concurrently active, the architecture relies on the use of local register values to maintain the power bandwidth balance.

[2]A conflict on memory read will result in a *bubble* in the pipe, a conflict on write will result in buffering in a vector register.

Performance Measurement

- Scalar Machines

  Given that an average single instruction takes

  $$t_1[\mu s]$$

  then we can say that the performance of the machine is

  $$1/t_1[Mips]$$

- Vector Machines

  The above assertion is no longer true since the vector machine may be executing several instructions in parallel.

  Ideally we might hope for a performance of

  $$p/t_1[Mips]$$

  where p is the number of operations carried out in parallel.

- Pipeline performance

  Consider an l stage pipeline performing an operation on a vector of length n. The time taken for each stage is τ (normally 1 clock cycle). There is also an additional set-up time $s\tau$ to prepare the pipe for calculation.

  Thus the time taken to produce the first result is

$$t_1 = [s + l]\tau$$

  then at a rate of one result per cycle giving

$$t_{pipe} = [s + l + (n - 1)]\tau$$

  or

$$t_{pipe} = t_0 + n\tau$$

  where t_0 is the perceived start-up time ($[s + l - 1]\tau$).

Performance Measurement

- ## Asymptotic Performance, r_∞

  The asymptotic rate is a useful measure of parallel computer performance, it indicates the rate of operations for a very large vector.

  $$r_\infty \equiv \lim_{n \to \infty} n/t$$

  Thus for our simple pipeline

  $$r_{\infty pipe} = \tau^{-1}$$

- ## Half–performance Length, $n_{1/2}$

  The performance of our computer with shorter vectors is indicated by the half–performance length; the length of vector that will result in half the asymptotic performance.

  $$r_{n_{1/2}} \equiv r_\infty/2$$

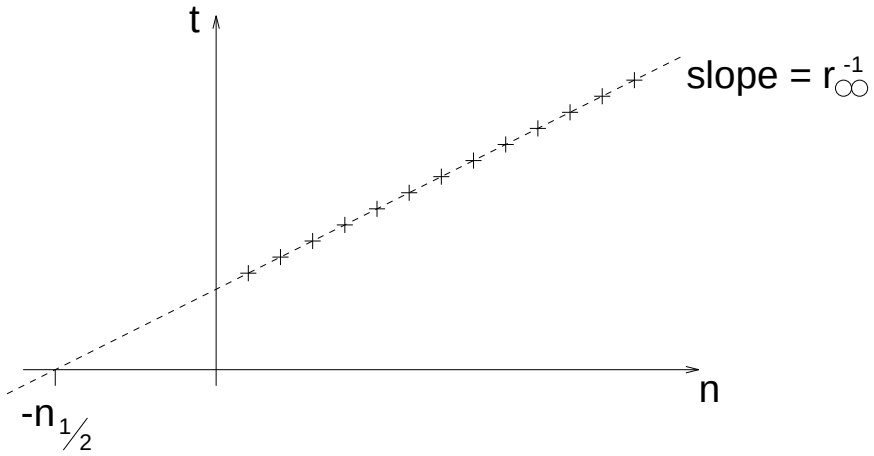
  Thus for our simple pipeline

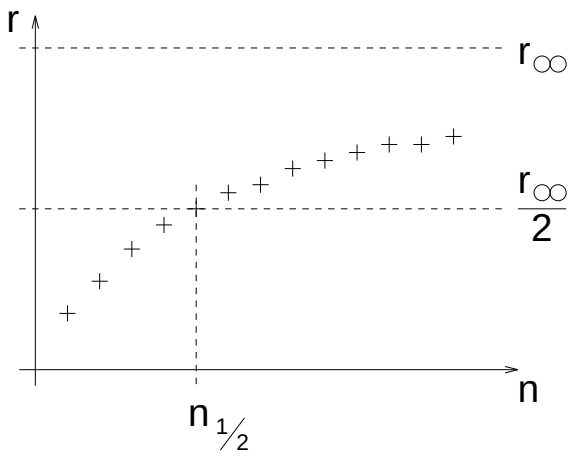
  $$n_{1/2} = t_0/\tau$$

Performance Measurement

The performance of our generic machine is fully defined by r_∞ and $n_{1/2}$.

$$t = \frac{n_{1/2} + n}{r_\infty}$$

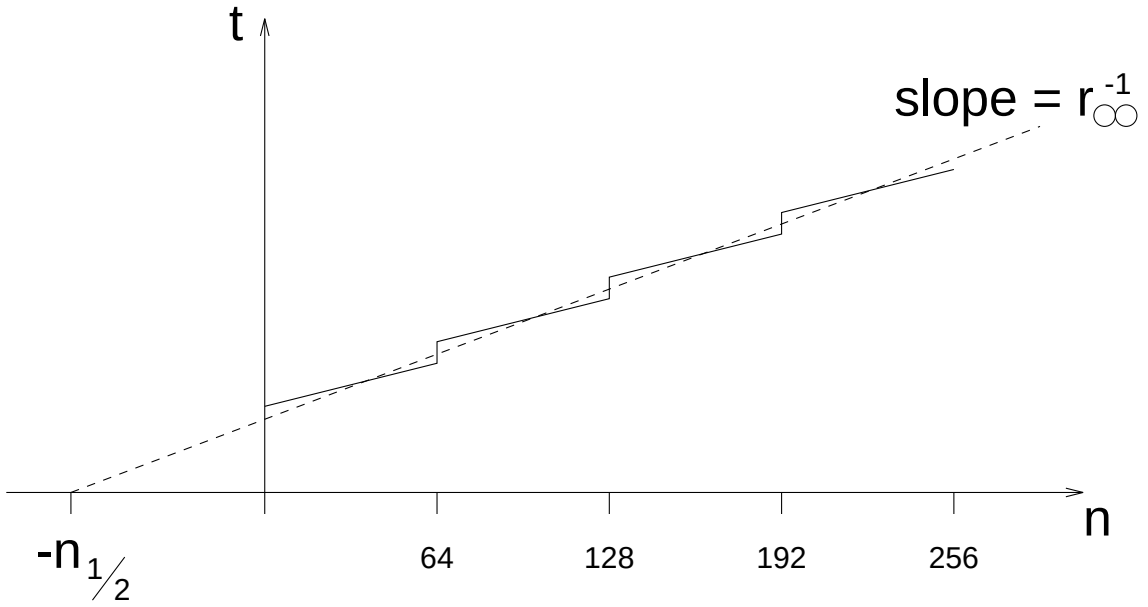




For most real machines this will not be the case, but it is usually possible to approximate their behaviour to that of a generic machine.

Performance Measurement

In the case of the Cray X-MP the performance will be modified due to the maximum vector length (64). This will result in an additional overhead every time a new internal vector instruction is started.



The solid line indicates the actual performance while the dashed line indicates the equivalent generic performance, a best fit line giving values for r_∞ and $n_{1/2}$.