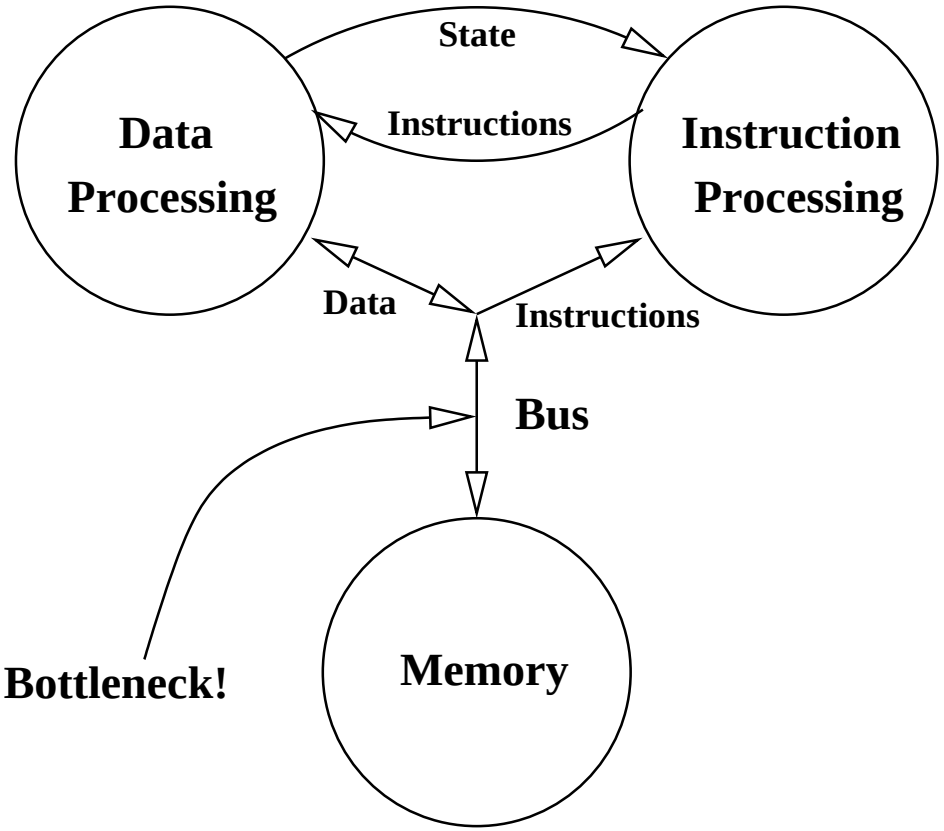
Pipeline Optimization

Dinesh Sharma

Microelectronics Group, EE Department
IIT Bombay, Mumbai

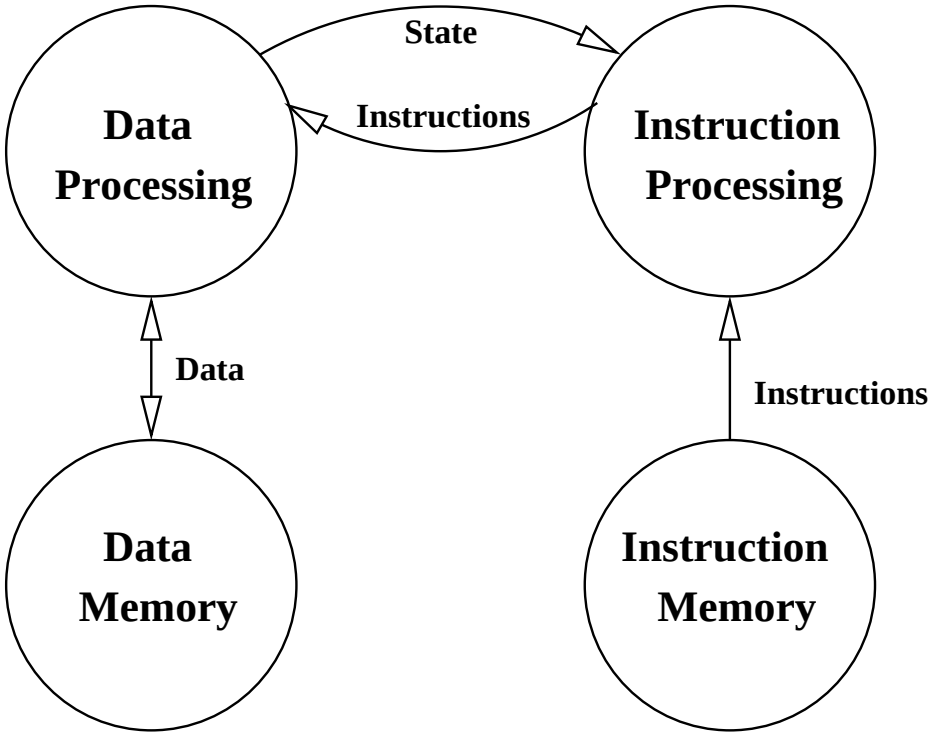
2006

Von Neumann Architecture



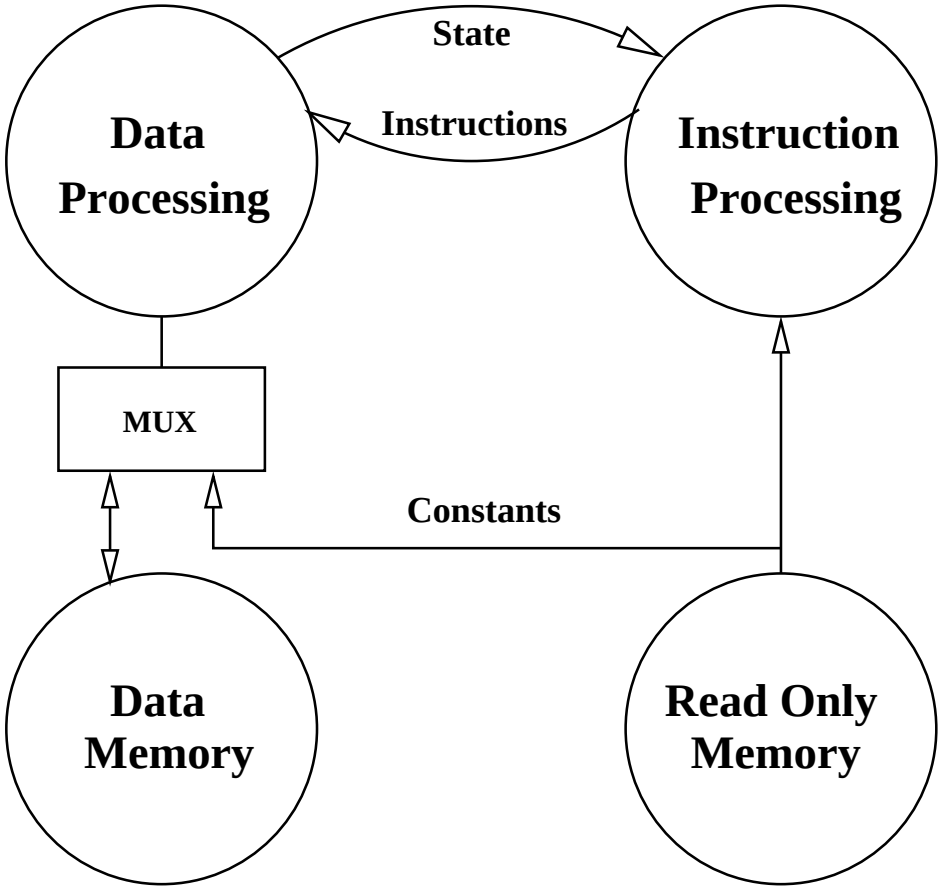
- A common bus is used for data as well as instructions.
- The system can become ‘bus bound’.

Harvard Architecture



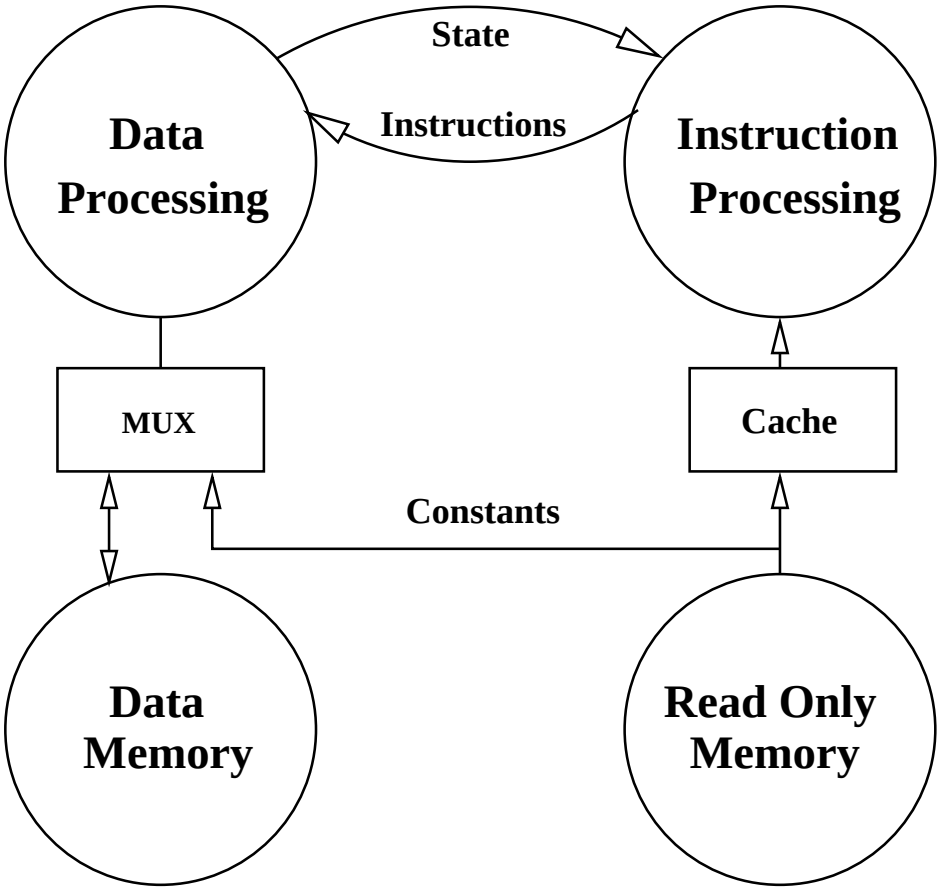
- Separate data and instruction paths
- Good performance
- Needs 2 buses $\rightarrow$ expensive!
- Traffic on the buses is not balanced.
- Instruction bus may remain idle.

Modified Harvard Architecture



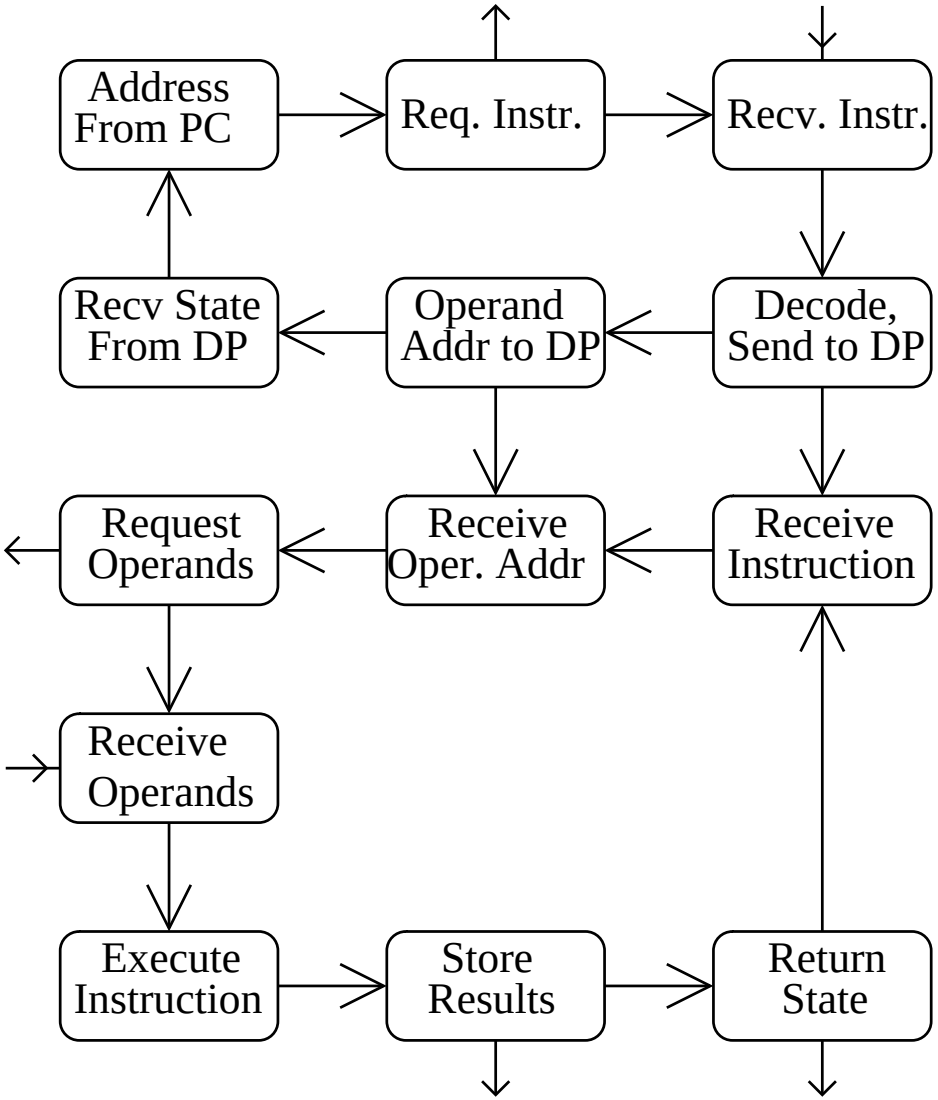
- Constants can be stored with Instructions in ROM.
- Better Bus balancing is possible.
- Typically, 1 instruction read, 1 constant read, 1 data read and 1 result write per instruction.
- 2 mem ops per bus.

Modified Harvard with Cache



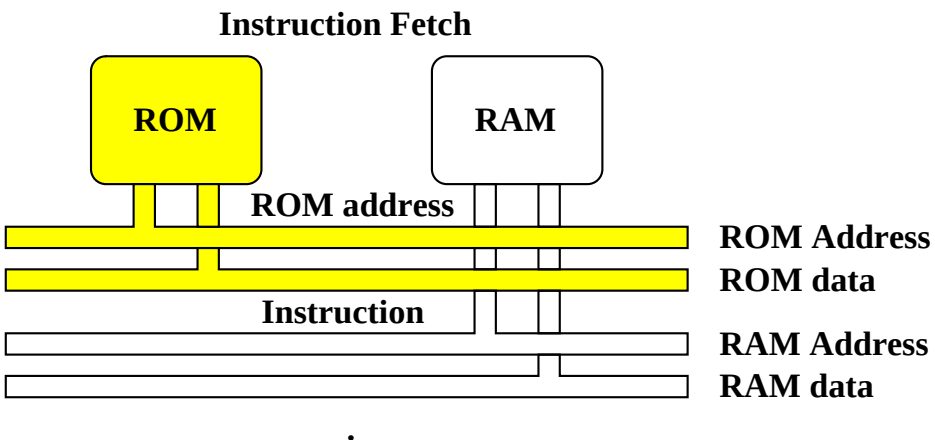
- Cache allows optimum utilization of bus bandwidths.
- Each operation need not be balanced individually.

Instruction and Data State Machines



- Operation of the system may be modeled as two interacting state machines.
- Instruction processor fetches instr, decodes and gives operation type and operand locations to data processor.
- Data processor fetches operands, performs operation and writes back the result.

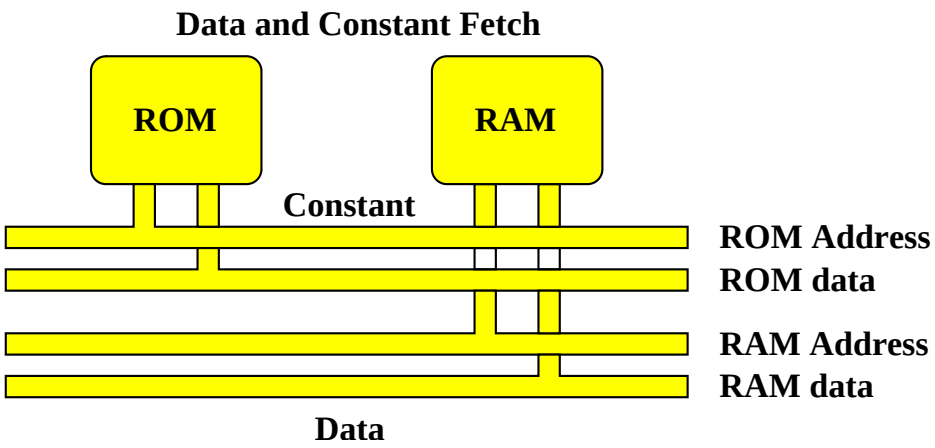
A pipelined processor



Consider a Harvard architecture processor, which performs the following tasks repetitively:

Fetch Op Code (ROM)

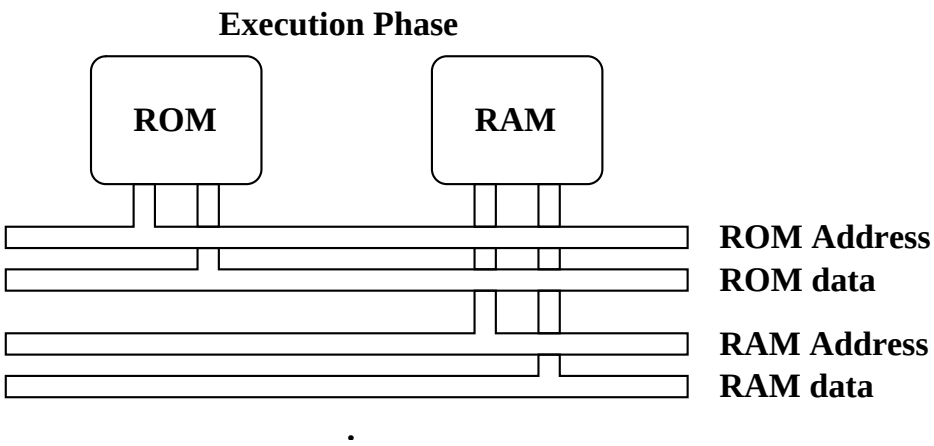
A pipelined processor



Consider a Harvard architecture processor, which performs the following tasks repetitively:

- Fetch Op Code (ROM)
- **Fetch variable (RAM)**
- **Fetch constant (ROM)**

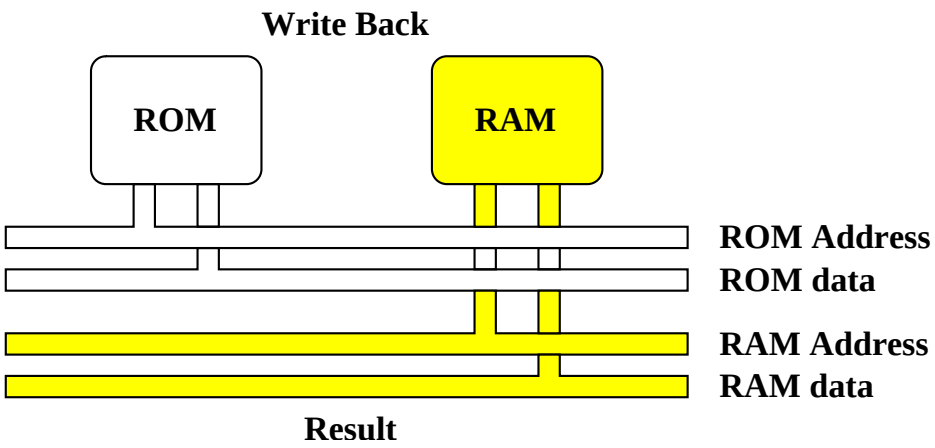
A pipelined processor



Consider a Harvard architecture processor, which performs the following tasks repetitively:

- Fetch Op Code (ROM)
- Fetch variable (RAM)
- Fetch constant (ROM)
- **Calculate result**

A pipelined processor



Consider a Harvard architecture processor, which performs the following tasks repetitively:

- Fetch Op Code (ROM)
- Fetch variable (RAM)
- Fetch constant (ROM)
- Calculate result
- **Store result (RAM)**

Resource Reservation

We can keep track of which resource is doing what at any given time by a table as shown below:

Resource Reservation Table

	0	1	2	3	4
ROM	Instr Fetch	Const. fetch			
RAM		Var. Fetch		Write Back	
ALU			Compute		

This is called a reservation table.
Given this reservation table, It appears that we can launch a new instruction every 4 cycles.

Overlapping Operations

However, we need not wait for the previous operation to be over before launching a new one.

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	0									
RAM		0		0							
ALU			0								

When can we launch the next calculation?

Pipelining

We can fetch the next instruction from ROM
while we write back the result of the current one to the RAM.

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	0		1	1		2	2			
RAM		0		0	1		1	2		2	
ALU			0			1			2		

This will enable us to launch a new calculation every third cycle.

Overlapping Operations

Is this the best we can do?

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	0		1	1		2	2			
RAM		0		0	1		1	2		2	
ALU			0			1			2		

None of the resources are utilized 100% in this scheme.
The ROM and the RAM are busy for 2 out of 3 cycles, whereas the ALU is used for 1 cycle out of 3.

A new sample is handled every 3rd cycle now.
Can we get even better throughput?

Improved Scheduling

If we store the result in a local register for 1 cycle,
and write it to the RAM only in the 4th cycle, we get

Modified Resource Reservation Table

	0	1	2	3	4	5	6
ROM	0	0					
RAM		0			0		
ALU			0				
BUF				0			

By delaying the write back,
we can launch the next instruction earlier!

Improved Scheduling

If we store the result in a local register for 1 cycle, and write it to the RAM only in the 4th cycle, we get

Modified Resource Reservation Table

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	0	1	1	2	2	3	3	4	4	5
RAM		0		1	0	2	1	3	2	4	3
ALU			0		1		2		3		4
BUF				0		1		2		3	

We can now launch a new operation every 2nd cycle.

Can this be further improved?

Improved Scheduling

If we store the result in a local register for 1 cycle,
and write it to the RAM only in the 4th cycle, we get

Modified Resource Reservation Table

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	0	1	1	2	2	3	3	4	4	5
RAM		0		1	0	2	1	3	2	4	3
ALU			0		1		2		3		4
BUF				0		1		2		3	

The RAM and the ROM are now occupied
100% of the time, So the design is optimal
and the throughput cannot be improved any further.

How can we always find the optimum solution?

- Given a Resource Reservation Table, we would like to set up a **systematic** method which optimizes the throughput of the process using this table.
- For maximum throughput, we would like to launch new operations as frequently as possible.
- Thus, we want to minimize the time gap between launching two operations.
- This is called the **Sample Period** (SP).

What is the minimum possible value of SP?

The minimum Sampling Period

- Consider an operation in which the busiest resource is used for n cycles.
- If we launch a new operation every n cycles, this resource will be used 100% of the time.
- If we launch operations any more frequently than this, the resource will not have enough time to do its work.
- Therefore, the minimum possible Sample Period is equal to the maximum number of cycles for which the busiest of the resource(s) is in operation.

Sampling Period

- We want to minimize the sampling period.
- But the sampling period need not be a constant!
- SP can cycle through a finite set of values.
- We should therefore define an **Average** Sampling period ASP.
- The minimum value of this **average** Sampling Period (MASP) is given by the number of cycles for which the busiest resource is used in an operation.

Cyclic Sampling Period

Consider the following reservation table:

	0	1	2	3	4	5	6	7	8
RSC1	0		0						
RSC2		0		0					
RSC3			0						

Now the next operation can be launched in cycle 1 itself. However, the following one can only be launched after a gap of 3 cycles in cycle 4.

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	1	0	1	2	3	2	3	4	5	4
RAM		0	1	0	1	2	3	2	3	4	5
ALU			0	1			2	3			4

Again, the next operation can be launched in the next cycle (in cycle 5) and after that, with a gap of 3 cycles in cycle 8.

Average Sampling Period

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	1	0	1	2	3	2	3	4	5	4
RAM		0	1	0	1	2	3	2	3	4	5
ALU			0	1			2	3			4

- New operations can be launched in clock periods 0,1,4,5,8,9 . . . .
- Thus, the sample period cycles through the values {1,3}.
- The average of the cycle is called the Average Sampling Period (ASP).
- The Average Sampling period (ASP) is 2 here.
- The whole pattern repeats every 4 cycles. This is called the period (p).

Minimum Average Sampling Period

- The minimum value of the Average Sampling Period (MASP) is given by the maximum number of cycles for which a resource is busy during an operation.
- Therefore, given a reservation table, MASP is known.
- If the actual average Sampling Period is equal to MASP, the system is already optimum and nothing needs to be done.
- If the actual average Sampling Period is greater than MASP, we can attempt to modify the reservation table, such that MASP is achieved.

Pipeline Optimization

1. For a given reservation table, find the current average sample period (ASP).
2. Find the largest no. of cycles for which a resource is busy.
3. This is equal to the Minimum possible Average Sampling Time (MASP).
4. If ASP = MASP, there is nothing to be done.
5. Else, we should try to re-schedule events such that MASP is achieved.

Method to achieve MASP

- We first consider various cycles whose average is the desired MASP.
- For example, if MASP is 2, we can have cycles of {2}, {1,3} or {1,1,4} etc.
- The periods are 2, 4 and 6 in these three cases.

The Generator Set

- For each cycle, we construct a generator set G, which contains elements of the cycle, their sums taken two at a time, three at a time etc., modulo periodicity p.
- In our example, cycles are {2}, {1,3} and {1,1,4}
  For a cycle of {2}, p = 2, so G = {0}
  For a cycle of {1,3}, p = 4, so G = {0,1,3}
  For a cycle of {1,1,4}, p = 6, so G = {0,1,2,4,5}

The Source Set

- For each selected cycle, We now construct the Source set S. This contains integers 0 through p-1, from which all members of G except 0 have been removed.
- In our example, cycles are {2}, {1,3} and {1,1,4}

Cycle	p	G	S
{2},	2	{0}	{0,1}
{1,3},	4	{0,1,3}	{0,2}
{1,1,4},	6	{0,1,2,4,5}	{0,3}

Design Sets

- For each selected cycle, We construct Design sets D_i which have the property that:
  if $a \in D$ and $b \in D$
  then $|a - b|$ also $\in D$.
- In our example,

Cycle	p	S	D sets
{2},	2	{0,1}	{0}, {1} and {0,1}
{1,3},	4	{0,2}	{0}, {2}, {0,2}
{1,1,4},	6	{0,3}	{0}, {3}, {0,3}

- Notice that Design sets do not depend on the reservation table.
- The sets G, S and D_i are constructed from the repetition cycles whose average value is the MASP.
- Therefore we can make a library of these in advance for different combinations of MASP values and cycles - and use them when needed.

Row Vectors

- We construct a row vector for each resource in the reservation table.
- The row vector is a set which contains the clock period in which a specific resource is busy.

Resource Reservation Table

	0	1	2	3
ROM	0	0		
RAM		0		0
ALU			0	

In this example, the row vector for ROM is {0,1}, for RAM is {1,3} and for ALU is {2}.

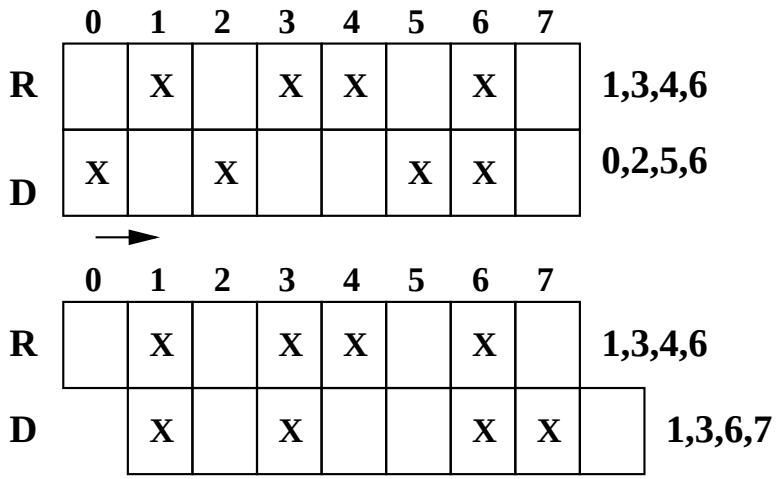
Matching Rows with Design Sets

- Choose a particular cycle with the desired MASP.
  (Say MASP = 2, cycle = {2}).
- Pick the corresponding design sets.
  (In this example, D = {0}, {1}, {0,1}).
- For each resource,
  take its row vector and take a design set with the same cardinality.
- Align these according to defined rules.

Rules for Alignment of the First elements

- Compare R(1) and D(1).
  If these are equal, nothing needs to be done.
- Else,
  - If R(1) $<$ D(1), add D(1)-R(1) to all members of R
  - If R(1) $>$ D(1), add R(1)-D(1) to all members of D
- This is equivalent to a rigid shift of R or D till their first members are aligned.

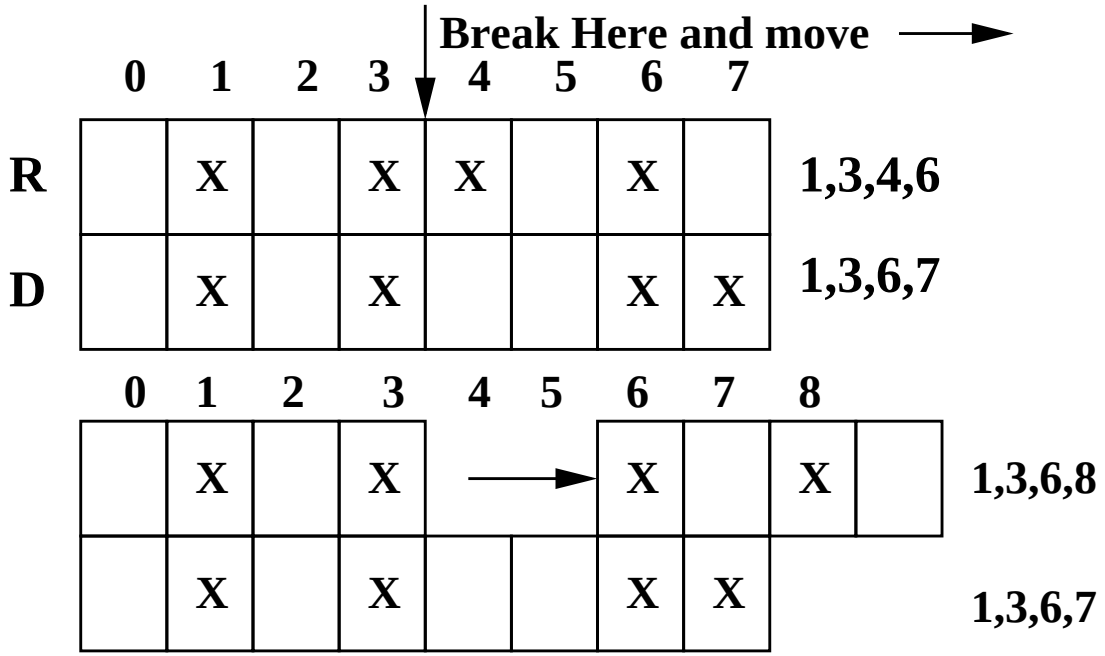
For Example, if R = {1,3,4,6} and D={0,2,5,6}



Alignment of other elements

If R(i) = D(i)] Nothing needs to be done.

If R(i) $<$ D(i)
Add D(i) - R(i) delays to all members of R at position i and beyond.

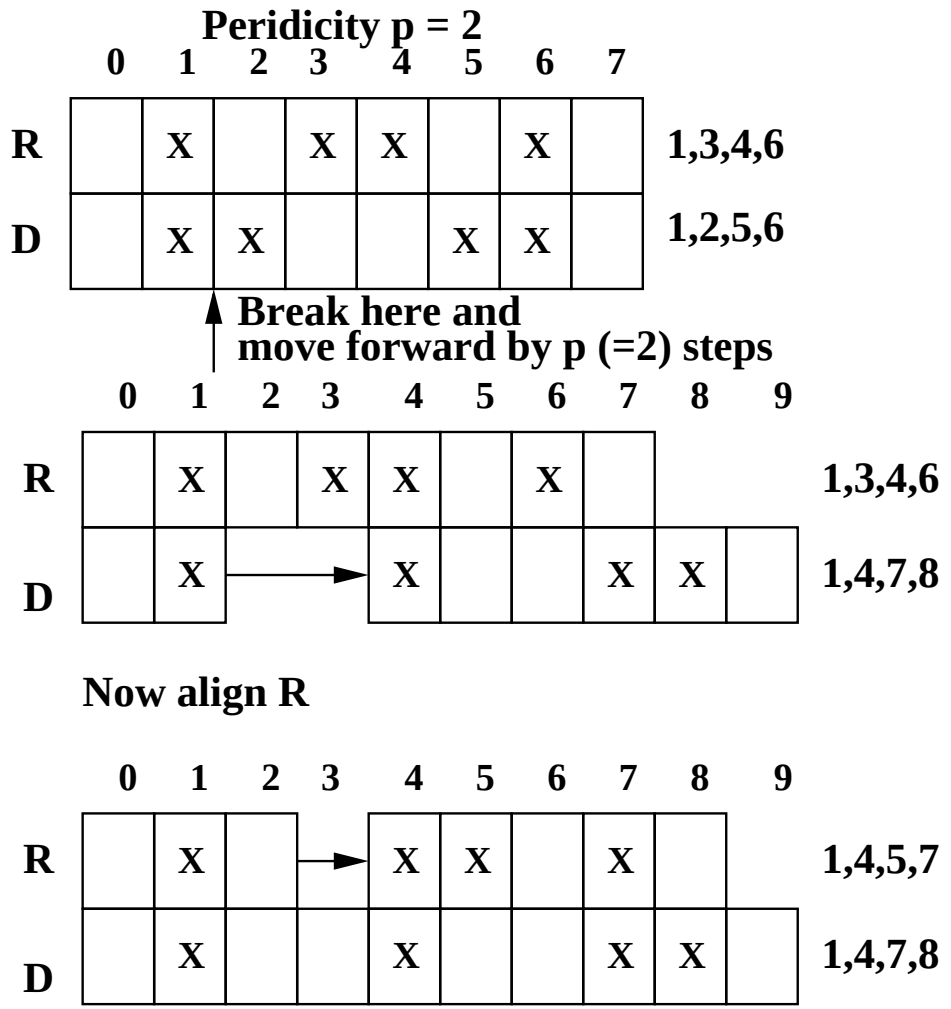


The i'th elements are now aligned.

Alignment of other elements

If D(i) $<$ R(i)
(for Example, p = 2
R = {1,3,4,6}, D = {1,2,5,6}.
Now D2 $<$ R2)

1. Add sufficient multiples of p to D(i) such that it is $\geq$ R(i).
2. Add the same number to members of D beyond i.
3. Now if R(i) $<$ D(i), add D(i) - R(i) delays to all members of R at position i and beyond.



Alignment Example

Let R = 1,3,4,6 and D = 0,1,4,5; with periodicity p = 2

	0	1	2	3	4	5	6	7	8
R		X		X	X		X		
D	X	X			X	X			
		X	X			X	X		

To align the first element, move all elements of D forward by 1 step.
Now D = 1,2,5,6.

	0	1	2	3	4	5	6	7	8
R		X		X	X		X		
D		X	X			X	X		
		X			X			X	X
R		X			X	X		X	

For the second element, D is behind. Move D2 onwards fwd by p = 2, so D = 1,4,7,8.
Move R2 onwards fwd by 1
So R = 1,4,5,7

Alignment Example

R = 1,4,5,7 and D = 1,4,7,8. R3 $<$ D3

	0	1	2	3	4	5	6	7	8	9	10
D		X			X			X	X		
R		X			X	X		X			
		X			X			X		X	

Move R3 and beyond forward by 2
So R = 1,4,7,9 and D = 1,4,7,8.
D4 $<$ R4
Move D4 forward by 2 to 10.
Now R4 $<$ D4.
Move R4 forward by 1 to 10

	0	1	2	3	4	5	6	7	8	9	10
R		X			X			X		X	
D		X			X			X	X		
D		X			X			X			X
R		X			X			X			X

Vectors are now aligned at 1,4,7,10.

Example System

we shall illustrate the method using our original example, whose reservation table is:

Resource Reservation Table

	0	1	2	3	4	5	6
ROM	0	0					
RAM		0		0			
ALU			0				

Since the ROM and the RAM are used for 2 cycles each in every operation, MASP = 2.
However, as we had seen before, ASP = 3 in this case.
Therefore, the schedule needs improvement.

Example Application

Aligning the ROM

	0	1	2	3
ROM	0	0		
RAM		0		0
ALU			0	

MASP = 2, Choose the cycle:{2}
Then D = {0}, {1}, {0,1}
For ROM: R = {0,1}, D={0,1}

So no alignment is required.

Adjusting the RAM Schedule

For RAM: R = {1,3}, D={0,1}

Aligning the First Element:	
R(1) $>$ D(1) Add (1-0)=1 to D elements	$\Rightarrow$ D = {1,2}
Aligning other elements:	
R(2) $>$ D(2) Add p (=2) to D(2)	$\Rightarrow$ D = {1, 4}
Now R(2) $<$ D(2) Add (3-2)=1 to R(2)	$\Rightarrow$ R = {1, 4}
R and D are now aligned.	

ALU Schedule

For ALU: R = {2}, D = {0}

Aligning first element: Add (2-0) = 2 to D $\Rightarrow$ D = {2}
R and D are now aligned.

ROM = {0,1}, RAM = {1,4}, ALU = {2}

Modified Reservation Table

	0	1	2	3	4
ROM	0	0			
RAM		0			0
ALU			0		

As we have seen earlier, this is indeed the optimal schedule with ASP = 2.

Optimized Reservation Table

Modified Resource Reservation Table

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	0	1	1	2	2	3	3	4	4	5
RAM		0		1	0	2	1	3	2	4	3
ALU			0		1		2		3		4

- The ALU is idle 50% of the time.
- Rather than buffering its result to delay the write back, we can use a slower ALU which takes 2 cycles to compute.

Using a Slower ALU

The reservation table with a slower ALU is:

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	0	1	1	2	2	3	3	4	4	5
RAM		0		1	0	2	1	3	2	4	3
ALU			0	0	1	1	2	2	3	3	4

- One can trade off power for speed when designing the ALU.
- By using optimization techniques, we are able to reach a higher throughput, even with a slower ALU!

Alternative Choice of Cycle

	0	1	2	3
ROM	0	0		
RAM		0		0
ALU			0	

MASP = 2, Choose the cycle:{1,3}
Then D = {0}, {2}, {0,2}

For ROM: R = {0,1}, D={0,2}
R(1) = D(1) = 0, R(2) < D(2)
Add D(2) - R(2) to all members of R at position 2 (and beyond)
$\Rightarrow$ R(2) = 2.
R and D are now aligned at {0,2}

Alternative Cycle:RAM Schedule

For RAM: R = {1,3}, D={0,2}

R(1) $>$ D(1)
Add (1-0)=1 to D elements: $\Rightarrow$ D = {1,3}
R and D are now aligned at {1,3}.

For ALU: R = {2}, D = {0}

Aligning first element: Add (2-0) = 2 to D $\Rightarrow$ D = {2}
R and D are now aligned at {2}.

	0	1	2	3
ROM	0		0	
RAM		0		0
ALU			0	

Time Ordering

	0	1	2	3	4	5	6	7	8	9	10
ROM	0	1	0	1	2	3	2	3	4	5	4
RAM		0	1	0	1	2	3	2	3	4	5
ALU			0	1			2	3			4

- As expected, the schedule is optimum.
- The sampling rate alternates between 1 and 3.
- **However** this schedule does not preserve time order.
- It asks for computation and constant fetch in the same cycle.
- If we pre-fetch the constant for the *next to next* calculation in this cycle and store it for 4 cycles, it may still work.

Conclusions

- Pipeline can improve throughput of systems.
- A systematic procedure for optimizing pipeline throughput exists. It can create modified reservation tables which are optimal by delaying some operations.
- However, it does not guarantee that the time order of different operations will be preserved.
- Different cycles with the same Average Sampling Period may have to be tried before an acceptable time order is found.
- The procedure also allows us to identify non-critical components which can then be redesigned to be slower but at lower power consumption.