

Computer Architecture
Prof. Mainak Chaudhuri
Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Lecture - 10
Basic pipelining, branch prediction


With gradual improvement, today and perhaps tomorrow whatever we will talk about should hopefully be recapped, some portions of 220, it is basically about pipeline, it is basics of pipeline. Keep in mind that essentially you will try to implement the instructions here that we discuss and you will be focusing those three on this.

(Refer Slide Time: 00:48)

Primer on pipelining

- Consider the cubing function: $f(x) = x^3$
 - We can write $f(x) = M(M(x, x), x)$ where $M(x, y) = x * y$
 - This decomposition of f allows us to implement f with two serially connected multipliers

- Suppose the multiplier latency is m ns
 - How frequently can a new input be sent to this hardware?
 - How frequently should the output be sampled?

 MAINAK CS422 2

So, it starts with a very simple operation rather a function in computation a function which is a cubic function. So, we will have to compute x cube minus, so we will try to see how we can implement the function. So, we can write $f(x)$ as M of M where M of x y is x times y that is you are cubing, so this decomposition of f allows us to implement f with two serially connected multipliers. So, this is how it looks like, so first you apply M and the output of it, you apply immediately, so this is done.

Suppose, the multiplier latency is M nano second, then the question aligned is how

frequently can a input can a new input be sent to this hardware, how frequently should the output be sampled. So, keep in mind that this, this is a purely combinational subject, there is there is no flop nothing when you leave x after some time comes out $f x$. However, it is very important to realize that the real point there is nothing that is close to this combination because a hardware that is useful will be used repetitively.

It will be used and that is also means that I give an input I give an output $f x$ and then we will give one more input one more output $f x$ give one more input give one more output $f x$. Immediately, when we start doing that these two questions become very relevant that how fast can I use this question use this multipliers to the hardware. I give x , then I give the next x in one micro second time, so that is a very useful question.

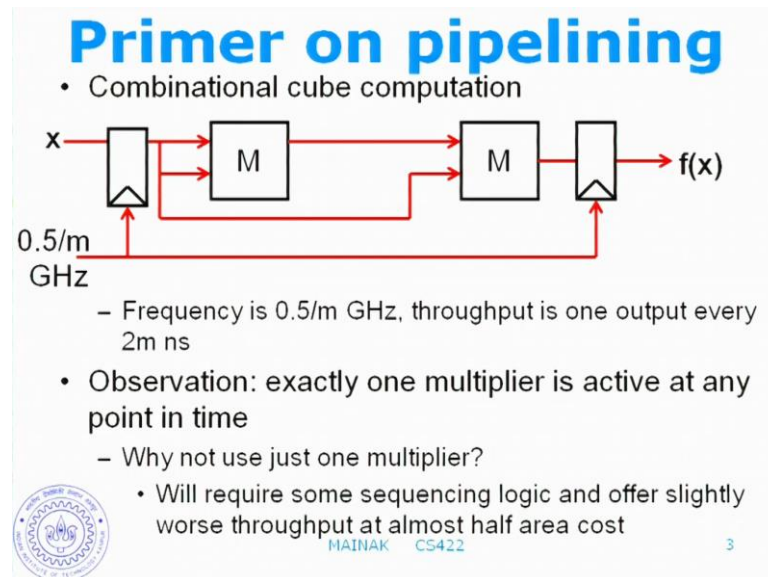
So, in this case what is the answer to this 2 m, so I give x output comes out after 2 M nano second and without any difficulty I should be able to give one more x once the previous x has already been computed. So, in this case I should be providing inputs at a rate slower than or equal to 2 M nano second, so one per 2 M nano second; however, and I should be sampling mine output also at the same way. So, what happens if I if I provide and do it in a faster way, suppose I give x and after less than 2 M nano second, i provide the second x , what will happen?

What will happen is will happen entire result, so what will be the result will it be something garbed, sorry. So, there is no legislature here, so this is a multiplier it takes x and gives new x square right and it takes x square and x give to x cube. So, I give x and give less than 2 M nano second, I will get my second x , what will happen to the computation what do we expect to see here what will you see here. There is nothing of value here, so what value do I get here, now see I give x and before I I reach 2 M nano second before I give the second x , what will happen here, what will you get.

So, it depends on when I start to do here and exactly how fast, what the parts are of this hardware, how fast these particular lines will be affected by this new input right. So, to be safe we have to obey these particular large at for these particular circuit I should not be providing inputs at a rate faster than 1 Giga Hertz. I should be sampling output at the same rate, so the thing I point here and then say is that there is nothing that is purely

combinational in here. There is nothing like that there has to be a sampling here somewhere, where you provide inputs sample outputs.

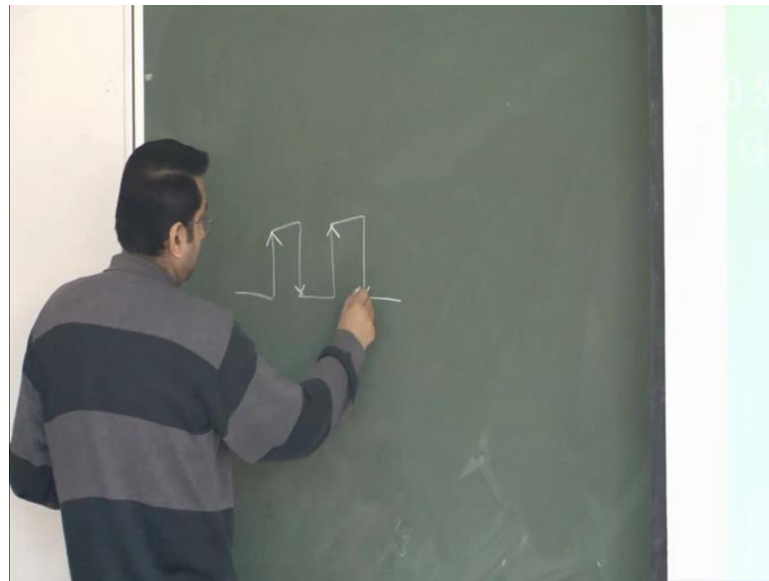
(Refer Slide Time: 05:57)



So, this is what it looks like these are some funny structures, but very important these are called what are they called sorry p flops, that is vulnerably realizes this, these are usually called latches, often good we call pipeline registers.

So, I will be using all these in your university in this course. So, the function of this is that very interesting, while this signal is present, it will take the value here and transfer that value to the output after some delay, so that is what it does. So, in this particular course we will be using one particular type of G h z that is we will assume that the signal given here is square root square root signal. This one is going to transfer the input to the output only if you are in a rising edge or on a falling edge that is called a exterior latch whenever the clocks, So, this is called a clock whenever the clock head strikes, it will take this input value here after a after a small delay that will appear here.

(Refer Slide Time: 07:24)

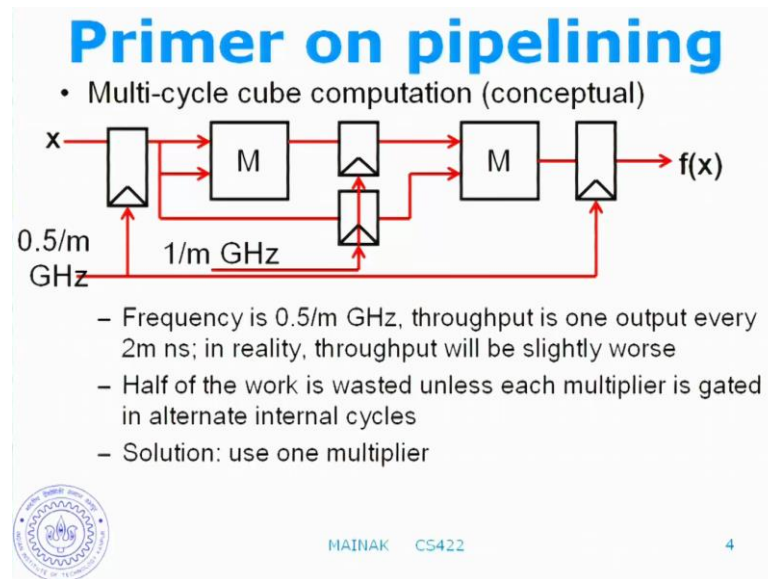


We will also distinguish between positive exterior latches and negative exterior latches. So, here at once that operate on the positive edge. So, these are the positive edges these are the negative edges, so this is my cubing combinational cubing circuits.

I will continue to call it combinational, you now know I have two latches separating these combinational circuit from the involvement. So, involvement is providing me x and giving out the affects to the involvement alright that I am doing that in a combinational fashion frequency is point five over M giga hertz that is throughput is one output every two M nano second any question so, far. So, one obvious observation that you can make here is exactly one multiplier is active at any point in time given input.

This one computes the square, then this one line this one this one computes the cube and the other. Then, when this one computes the square this one is the environment, so why not use this for multiplier obvious question. So, if you want to do that, it will require some sequencing in logic and offer slightly worse throughput at almost half area cost.

(Refer Slide Time: 08:44)



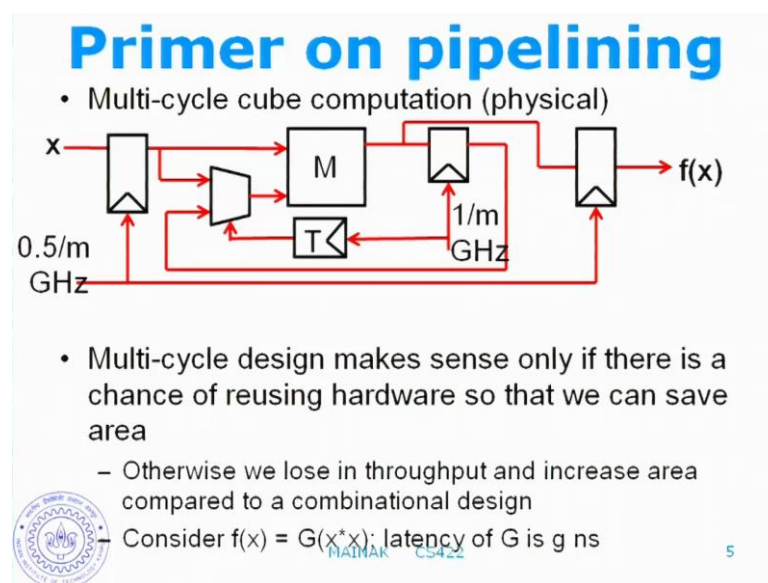
Let us see what that means, so conceptually, what we will really want to do is we will first use this multiplier to compute the square and you know I do not know you will remember the output of that and leading back to this. That is what you will ask me what to do, so first I am surely clear that the conceptual fashion that is by unfolding the whole thing, so but we still have two multipliers, but I will introduce these, now the remaining. So, that the pair of latches latching x and latching x square and these one has to be operated faster. It has to be operative in one more impulse it does this one is ready in M nano second.

So, I must take a clock and write it so that in the next M nano second it will be slightly involved alright. So, the frequency is seen by the involvement is still 0.5 over M Giga hertz alright throughput is one output every $2M$ nano second in reality throughput will be actually slightly worse, why is that why we have not gone there I am just talking about these, We have a latch in between, but I still have two multipliers, so I am thinking next ladder am going to fold the multipliers, but about this am just talking about this one. Now, why is it exactly, so that will stop the air to going through these preformation, so what will happen is that at every M nano second this clock is going to strike.

So, on the rising edge I will take whatever value is here, but after only some delay, this

value will be accurate here, so that too I am taking in account and continue to the latency of this subject. I will introduce otherwise it is all coming for free we are losing time here, so here still we waste half the work unless each multiplies get it alternate internal cycles. So, that is one option I can say that well at every internal every alternate internal cycle I will disable this multiplier if there is somewhere. So, there essentially that is going to sleep I will not waste any energy or power switching that multiplier. So, of course, the obvious solution is to fold them together, so let us see what that looks like what you will do, so the circuit gets a slightly more complicated.

(Refer Slide Time: 11:22)



So, let us see why that is, so essentially I have brought this multiplier and put it on that. So, the latch remains unchanged, here the only problem now is the quizol for the second input is going to be to this multiplier one input was x of course. So, if you look at the second multiplier, it was still taking x as one input. So, the other input was x square right and if you look at this multiplier both the inputs are x , so when you bring this here you will know the inputs correctly.

So, let us see how to do that input of multipliers here, so one input is x all the time, the other input switches between x square and x and this is a toggle latch, which will keep on toggling at every siren 0101. So, whenever this goes 0 it will pick up x whenever this

goes one it will pick up x_1 , so that is the sequencing logic that I was talking about. So, otherwise everything else remains unchanged, the only thing that we have done here is that we have added some more latency in the circuit.

So, it is going to be even worse than the previous one, we have this latch now and we have this multiplier now looking into the mark this toggle latch is not exactly a critical ones. This multiplier will be seeing the critical part of the computation unless it is solved because of the second impression goes through the multiplier because eventually it will get it there alright multiplexer. So, this a multi cycle design that is what we can call it and the reason is obvious that I have a faster clock one over M Giga Hertz, but I will take multiple cycle of that clock to compute this output.

However, if u look at the involvement, the involvement still sees the same sample weight, it does not improve, it will happen if you have learned you have not done anything. You have just hold one multiplier on the other you are just using that multiplier twice, so the involvement still sees the same sample weight. The overall throughput will be now the self weight because, now the critical part will now take this multiplexer with this latch.

So, multi cycle design makes sense only if there is a chance of reusing hardware so that we can save area. So, keep that in mind otherwise there is no point in doing multi cycle design here, you could actually use one multiplier that is why you will do a multi cycle design. So, multi cycle design you will actually go to the multi multiple designs regarding the output, otherwise we lose in throughput. As you can see here, we lose in throughput and increase area compared to combinational design.

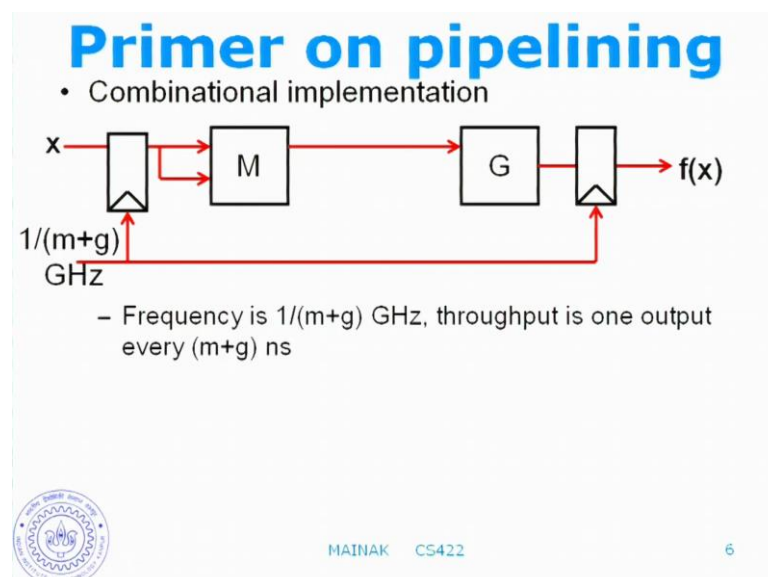
So, now you go back to the one that we started with this one this one still gives you the best throughput, actually after all this options that we learnt here till now, this will be very close to $2M$ alright and what I have done gradually is that I have reduced area, but I have increased square foot. I have increased it according to my latency to the cycle, so that is the circular point and coming right here. That you should be doing such an optimization only if there is a chance of winning in some department.

In this case, we will do chance of area on the subject, but we are losing in throughput compared to the combinational design. So, as a different example, suppose $f(x)$ is this G of x times x and let us suppose latency of G is G nano second.

So, before we move on to look at this particular function any question on this, what are the consequences. So, you are saying that this toggle latch and the multiplier is the controlled unit, but then you can make a logical division, but that is an obstruction. In reality, things will through all be together sitting side by side depending on also the one primary optimization role which can minimize the wired line because we have talked about that in one section that on access slope communication slope.

So, while exactly this will still be the fourth time, this will go somewhere else, you have a question on this multi cycle design and how does it function. So, let us take this one, now this is slightly different the difference comes, in the fact that it does not really use the multiplier. The second time it inputs the different function G the second time after using the multiplier.

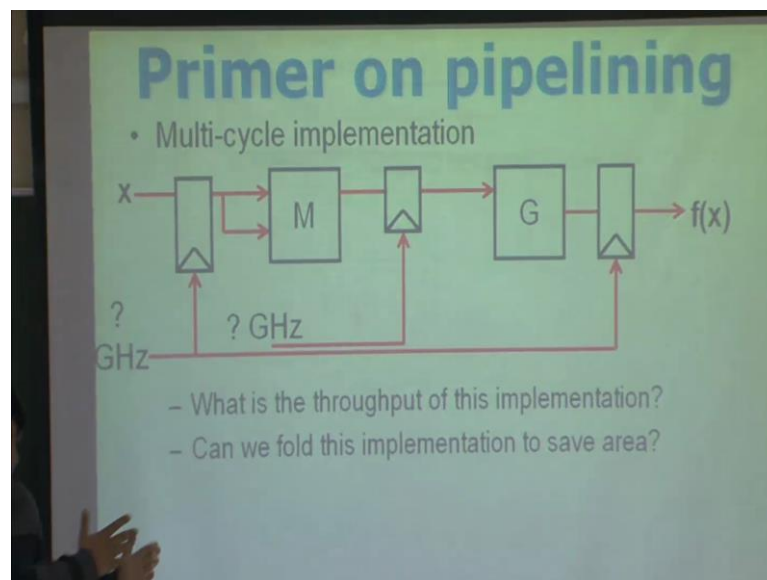
(Refer Slide Time: 16:36)



So, these are my combinational design my involvement has to sample at the rate of one over M plus G Giga Hertz alright to be correct. It should be faster than the sample rate

and throughput is going to be 1 output every $M + G$ nano second. So, now, if I ask you should I go for multi cycle design with this, what do you think, this is what it looks like multi cycle design.

(Refer Slide Time: 17:10)



I have an internal clock, I have an external clock and internally I initially latch this of course, I cannot point on to this because these two are different, so this is all I can get. So, now the question is what do I again by doing this, really I do not gain anything in terms of area, I am actually losing in terms of area because I have a basic to latch here. So, do I gain in terms of throughput, so how do you resolve these two question marks here, what are these frequencies start with the internal one, how fast should I clock this one what are the options do we have small M small G $M + G$, any other options.

So, he says $M + G$ both places $M + G$ are you sure initial shift, sorry what why is that shifted that is. So, you are worried about the fact that the first output where we have at fine good forget about that $M + G$ is that good, you think why is it $M + G$ this one what is the reason? So, if you go back to the previous one internal clock was $1 / (M + G)$ here, it was not $1 / (2M)$ and this is correct. So, do you just believe in what I am saying, you can argue that this is correct going to produce correct result, so I could do $1 / (2M)$ that will still be correct.

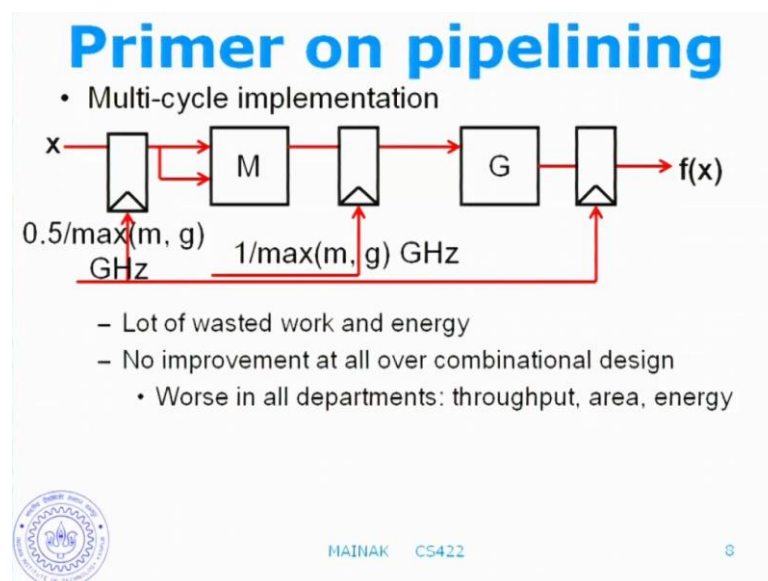
That will slow down your economy, so what should I do here, what is the right thing to do what is the fastest clock that I can feed here, what about m , why alright is that enough were it only about visualize the slower ones. So, this is going to be $1/\max(M, G)$, we should raise the slower one and the ratio of the sample output of this one here with this one G should be working with something. Otherwise, I will miss of that whatever computation was going on, what about this one, what is the involvement of this M plus G .

I sample it at $1/\max(M, G)$ am I good looking at how my M plus G sorry x plus m , see once you realize that these two clocks are actually tied together. They are depending on each other you cannot decide to manipulate it, you will get the answer immediately when you say this is $1/\max(M, G)$, how fast can you stop the involvement? I did not even see the problem with \max plus G .

Student: $1/\text{twice of } \max$.

$1/\text{twice of } \max$, what is the problem with these samples \max plus G exactly? So, there is a chance of over writing the computation actually, so these one actually has to be $1/\text{twice } \max$ of M plus G or M comma G , so this is what it looks like.

(Refer Slide time: 22:47)



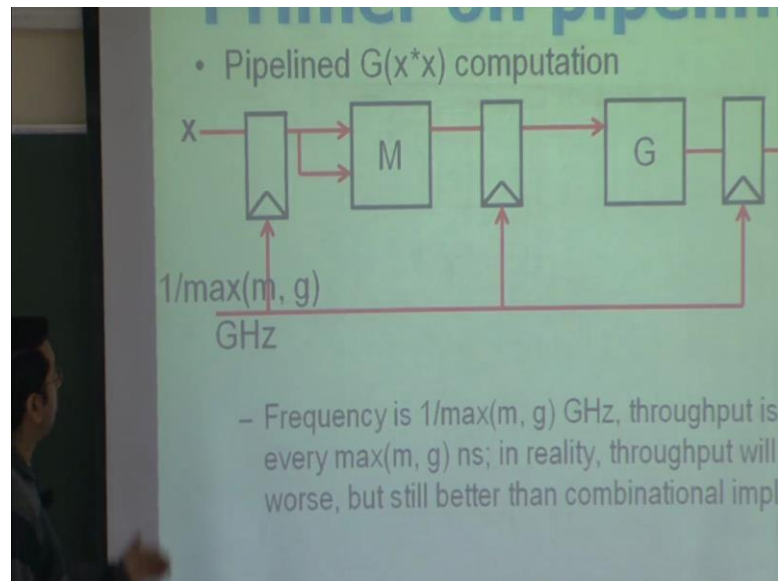
So, what have we gained by doing this? There is a lot of wasted work and energy because half the time we will be waiting these they include the all over combinational design. In fact, it is worse in all departments look at the throughput, it is going to be worse twice max of M plus G is going to be bigger than M plus G , then it is equal to M plus G . We have worse, we have added latches and energy that too will be worse because of the energy continuing with the latch. By effort, we get the latch the latches will be working together, so this is a typical scenario where we should not be going to a multi cycle design whatever we realize.

Now, moving on to the pipeline, so that is the next enhancement, so multi cycle design gave you age in terms of area, but you lost the proper throughput. So, now through new pipeline of course, we latch input the multiplier back, we need two multipliers. This is going to be one stage of the pipeline and this is going to be the second stage of the pipeline. So, now we add to this the involvement of the improvement that is the final home, I want the involvement to see the improvement this is the only case study you have to start doing the involvement improvement. So, now I can actually sample the input and the output at much faster 1 over M and because what I can do is that take the input feed this multiplier while this multiplier will be working on the previous one.

Then, throughput is 1 , output every M nano second, in reality here the throughput will be slightly worse because of this latch here. So, the fundamental requirement of pipelining is that you should be able to decompose the function to be computed into a series of functions. So, in other words you should be able to express the $f(x)$ as such a sequence of function, so at one stage I will do $f_k(x)$ in the next stage, I will do a f_k of k minus 1 of that and so on and so forth.

Ideally, one would expect a k times faster clock and k times higher throughput, the basic of pipeline. That is how you should start you can pipeline any computation, but the computation should be able to be broken down into such a point question, it will do yeah better, and it will keep on computing the same thing with them. So, as long as you sample at 1 over $2G$, however you are still going to collect the sample, so anything else you do, will make an impact any question pipeline.

(Refer Slide Time: 26:39)




So, how do you pipeline these one, so here this is how you pipeline this computation and involvement will also see the input immediately with $1/\max(M, G)$. So, that is definitely a combinational circuit in which you will see a throughput of $M + G$. $M + G$ is going to be less than or equal to $\max(M, G)$. So, frequency is $1/\max(M, G)$ Giga hertz throughput is 1 output to every $\max(M, G)$ nano second and again in reality the throughput will be slightly worse, but still better than the combinational implementation that is the whole point.

So, pipelining if done well should always be benefit over combinational implementation in terms of throughput. We are not commenting anything on area, because in most cases you would actually end up increasing the area because we will have to introduce latches and all the others any question?

(Refer Slide Time: 27:47)

Primer on pipelining

- Imagine feeding a series of inputs x to a k -stage pipelined implementation of some computation $f(x)$
 - Usually, one would expect the computation on two inputs to be independent of each other
 - True for stateless or memoryless functions
 - Consider the following function with a global state r
 $f(x)$ returns y
if $(r == 0)$ $y = x * x * x$
else $y = x + x + x$
 $r = y \bmod 3$
- How to pipeline this function?
 - Assume two multipliers and two adders
 - Computation of one input depends on the previous one
 - Pipeline hazard



MAINAK CS422 11

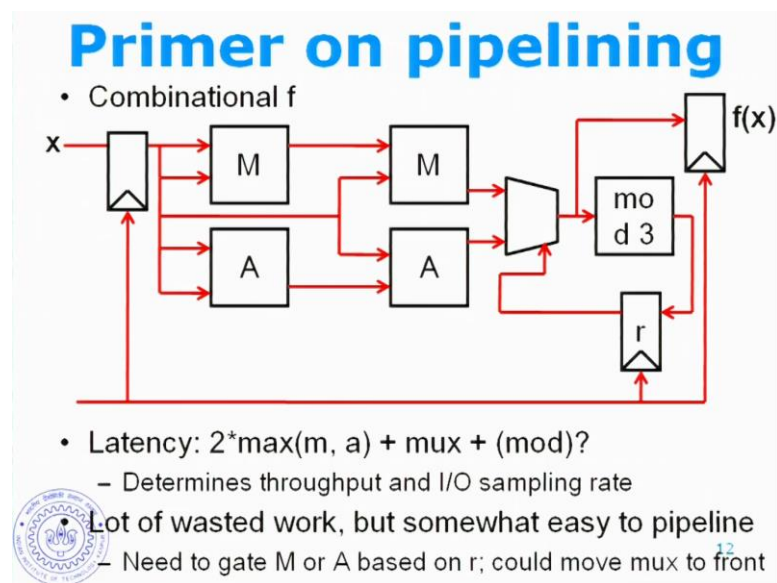
Now, let us go a little deeper into pipeline imagine feeding a series of inputs x to a k stage pipelined implementation of some computation of $f x$. So, usually one would expect the computation on two inputs to be independent of each other, if you feed x_1 , the pipeline x of x_1 computes $f x_1$ and next I give x_2 pipeline computes $f x_2$ of x_1 to or something 3 of $f x_3$.

So, here we should expect the computation of two inputs will be depending on each other this is true for very some class of functions which are called stateless or memory less functions. That is when you are computing f of x_1 , what about the stage, we produce forget them and that does not influence your computation of $f x_2$, so these are called stateless or memory less functions.

Consider the following function with a global state r , so now we start talking about the function which is state holding and let us see how pipelining gets complicated in dealing in such things. So, here the function differentiation $f x$ becomes y and so does $f x_2$ if r equal to 0, so there is a global state r if r is 0, then it computes y with x cube else it computes y with 3 x . It either multiplies the three values or adds G values and finally updates r equal to $y \bmod 3$.

So, now what do you think I keep x_1 , what I will do for x_2 is what we will know until I finish computing f of x_1 , so how to pipeline this function? So, we assume that we have two multipliers and two adders computation of all input depends on the previous one that is pretty obvious right because the value of r is going to change. So, these are called pipeline hazards that is now the two inputs cannot be computed independently, I cannot compute a particular value ignoring history of the pipeline that is impossible.

(Refer slide Time: 30:10)



So, let us see how we pipeline this one, so first I will start as usual with the combinational presentation of f . So, this one computes x square, this one computes $2x$ this one computes x cube, this one computes $3x$, then I take r bringing through a multiplexer. So, based on r , which one I should do and then apply this one to a mod 3 block to update r and this one will be your output y . So, what is the latency of this, its latency is going to be 2 of twice max M coma a , so these two are very concurrent plus the multipliers latency plus the modular latency.

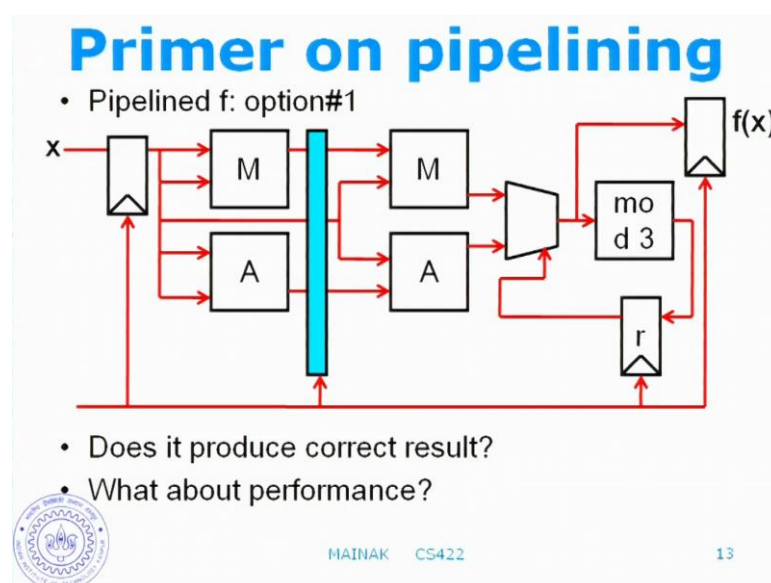
Now, there is a question here should I use this one to my latency or not because my output is actually ready at this point that is why the involvement gets stuck. What do you think should the modular latency should be inside or not and the latency is going to decide at what rate my input and output will be sampled? Actually, it should be why

exactly, but remember that am using r ones later, but of course, my modular function maybe so slow when the next input comes in, we will just computation and add the multiplexer.

I have not even updated r that would be impossible actually, so you always say you include this in my latency, so it determines the throughput of i o at sampling rate. So, as you can see here there is a lot of wasted work because you know I will be either taking this or this one, but am computing both, but am planning in this way because it is somewhat easy to pipeline. Otherwise, what you could do is you could gate M or a based on the value of r , it will disable which ever path you will disable and you could move the multiplexer to front and I could select which path to take.

So, that becomes very difficult to pipeline, we will see what requires exactly this case in pipelining of pipeline process. Here, the fundamental problem is that when x comes in, we do not know which way to go we do not know whether we should take this path or we should take this path until and unless you know the value of r . So, we will stick to this particular design, we will waste a lot of hardware, but it is easier to pipeline any question on this.

(Refer Slide Time: 33:05)



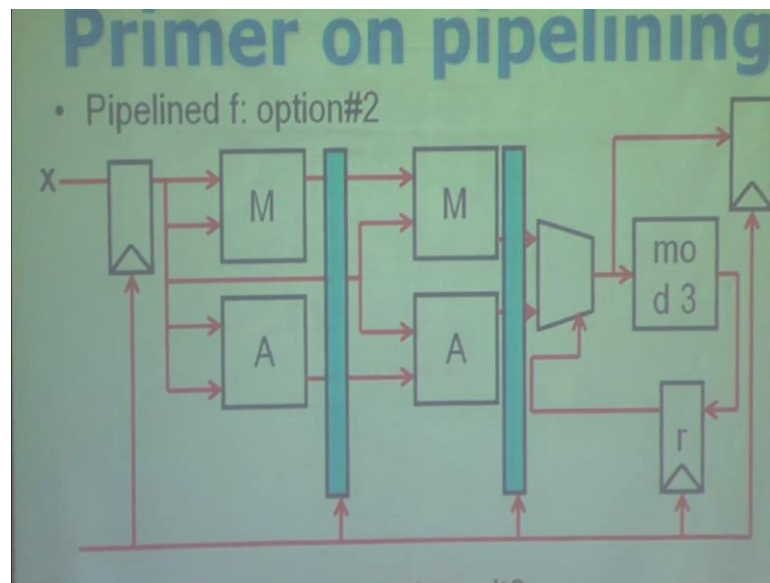
Let us see first option, let us suppose we want a two states pipeline, so into the pipeline latch here. So, first question that will be answered whether when you try to pipeline something, does it produce a correct result, what about the performance that is the first question what do you think? How to realize this particular circuit or are you thinking I assume that you will be thinking for sure. So, I will tell you the answer, yes it does because this one computes your x square and $2x$ that passes on to the next stage and I will be taking the next input here and there is no problem as such.

The only question was about r , but there is no issue with r because if you know this am clocking r with the same clock. So, this one gets modified only after I read out my output, which is the same clock. So, the r value that we modify in the multiplexer is the value that we had when the computing started there is no problem.

What about performance, how good is this pipeline, what is the clock rate of the of this pipeline, how fast can you run this pipeline which stage is going to be the pipeline to the frequency this one. Why is that and image coming at you have something more of extra here this is going to be the slower stage, so as we have already seen we should take the max.

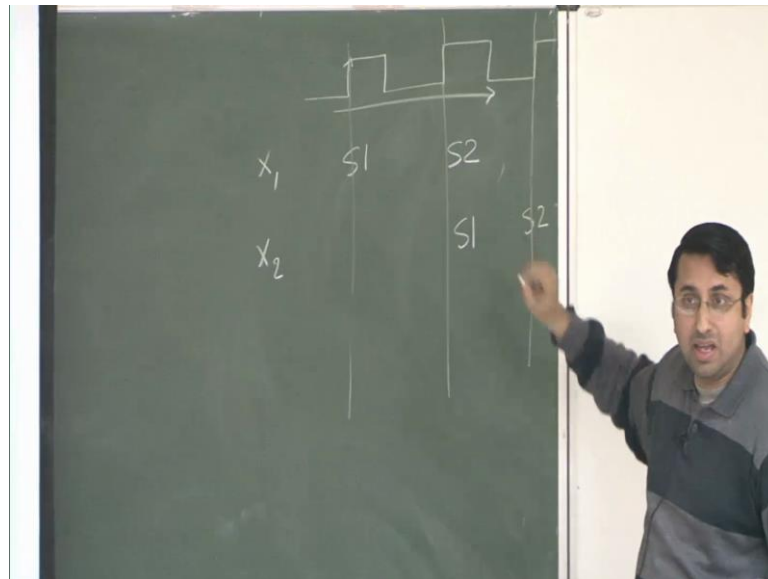
So, that means these are often called unbalanced pipelines, so we have two stages, which are not balanced pipelines. That means, there is a chance of improving because I should be able to if I can balance them, I must be able to gain in clock frequency right because the slower the slower stage will be my clock frequency.

(Refer Slide Time: 35:49)



I will make it a three stage pipeline, so again the same question passed will it be correct, that is still correct, because as I just mentioned the value of r that is the particular value of x is going to use is going to be correct. So, because r is going to modify in here and the value will get out through the input, but you have to be a little careful here. You are reading from and writing to the same register in the same pipe stage, which was true even for the previous one. So, we were writing to this particular register and reading from this register the same pipe stage.

(Refer slide Time: 36:46)

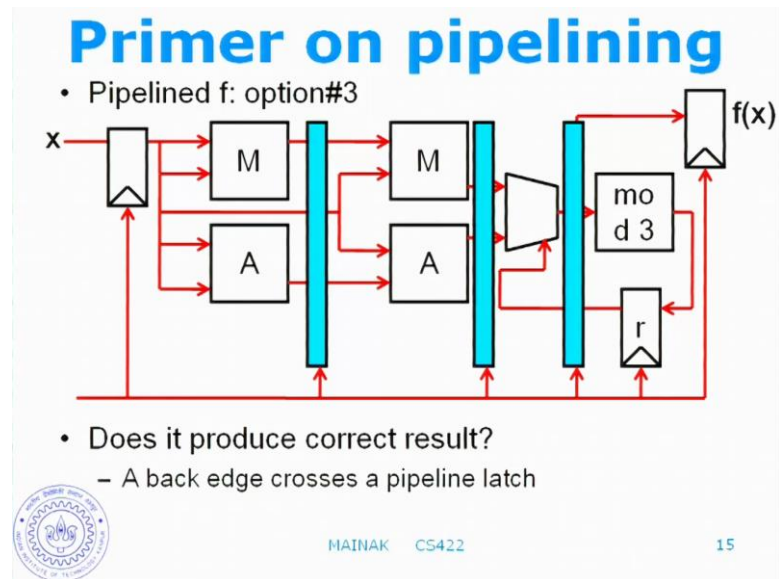


So, if you want to see what happens exactly, so let us call this s_1 and s_2 , two states, so the time goes in this direction? So, for a particular value of x , there is x_1 , next time I will observe this one and this too alright x_2 starts here. So, when s_1 is doing s_2 s_2 will be s_1 , then it does s_2 here, so there is a problem here because the value of r that s_1 is going to use here has already been modified, does everybody see that?

So, because what here is going to happen is that if you this is where x_1 gets sampled, this is where this latch will be sampled this is where the r value will be updated. This is exactly where x_2 s_2 will start executing, what about performance did it improve upon the previous pipeline, what is the what is the expected throughput of the of the clock frequency who determines the clock frequency of this pipeline max of all 3.

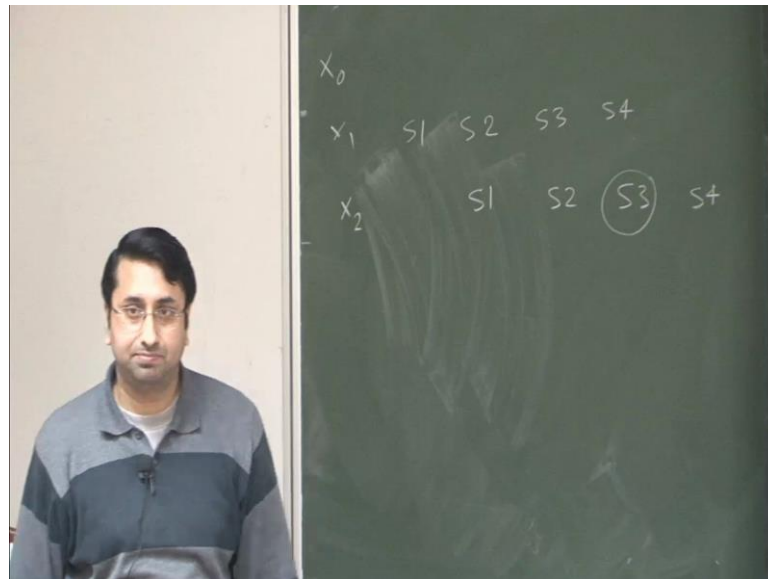
It is regularly going to be based on the previous one because the longest one in the previous case has been broken down to 2. Now, if all the algorithms are same, you can expect that either this or this will be the in determining stage, because these are going to be in a fast stage in the mod 3 operation and the multiplexer.

(Refer Slide Time: 39:06)



Suppose, I had a mod three implemented as a divide possible, actually that is going to be very sorrow. So, instead I will let you isolate that is this correct are you looking good yet or is there a problem with this one. So, there is a comment here a back edge crosses a pipeline latch what do you think of it what is the problem he says there is some problem with the r value what is the problem can you explain it more . There is a mildly case here I take you that it is wrong under all the influence, so if you try to do this again for this pipeline.

(Refer slide Time: 40:31)




So, we have s_1 , s_2 , s_3 , s_4 , x_1 , x_2 , so I need the value of r at s_3 stage, so s_2 gives value here the value's latched here after s_4 . So, there is no way you will get the right value, so x_2 will actually get the value of x not the previous one, which is wrong. So, keep this in mind whenever you see that in a design a value is getting produced in a stage, which comes after the stage that consumes the value.

You know that there is a problem this pipeline is not going to work again and that is which it translates to a back edge crossing the pipeline latch in a pan diagram. So, these are called pipeline hazards and resolving these ones is not going to be easy as all you need a value here which is latched here, in fact let us take it as a end of s_4 , I need the value of the beginning of s_3 , there is no solution.

(Refer Slide Time: 42:12)

Primer on pipelining

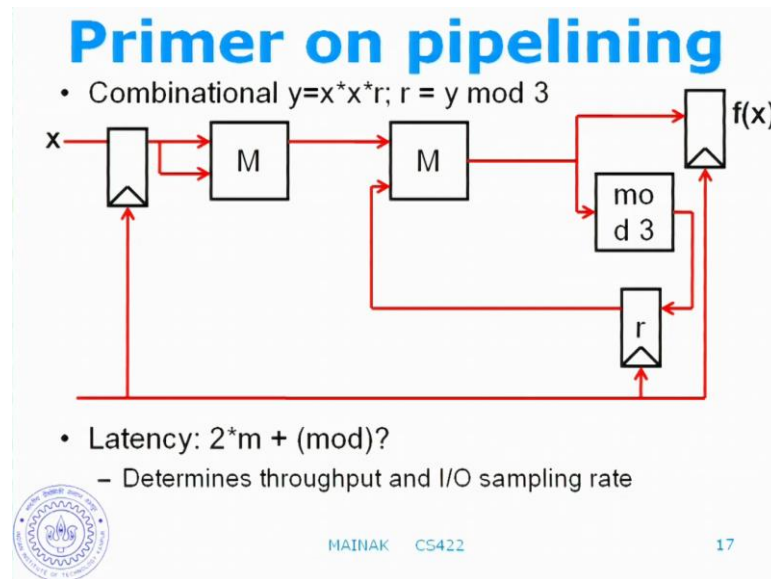
- Another hazard of different nature
 $f(x)$ returns y
 $y = x^2 \cdot r$
 $r = y \bmod 3$
- The nature of computation is known, but the required data may not be available
- Another similar example
 $f(x)$ returns y
 $y = x^2 \cdot r^2$
 $r = y \bmod 3$
- How to pipeline this one?



MAINAK CS422 16

Here is another hazard of slightly different in nature, so we are going to the hazard the problem was that given x we did not know what to do with this x which way to go and which way to go depends on the computation of the previous x . Here, we know what to do, but we have no data ready to do that, so here there is a function defines $f(x)$ returns y y is x square times r and r is $y \bmod 3$. So, here the nature of computation is known, but the required data may not be available, so we will we will try to try that and realize that. Another similar example y will make it x times r square has it to x square times r square and r is again $y \bmod 3$, so let us see how to modify these two.

(Refer Slide Time: 43:03)



So, this is y is x square times r , so again we will start with the combinational circuit, so you have two multipliers the input from r the second multiplier and the remaining things are same, latency is $2M$ plus \bmod . You know the modular latency that determines the throughput and i o sample rate, so I put the pipeline latch there, are we good? Yes we are good, there is a problem where r is consumed and sorry M is consumed in the same five stage.

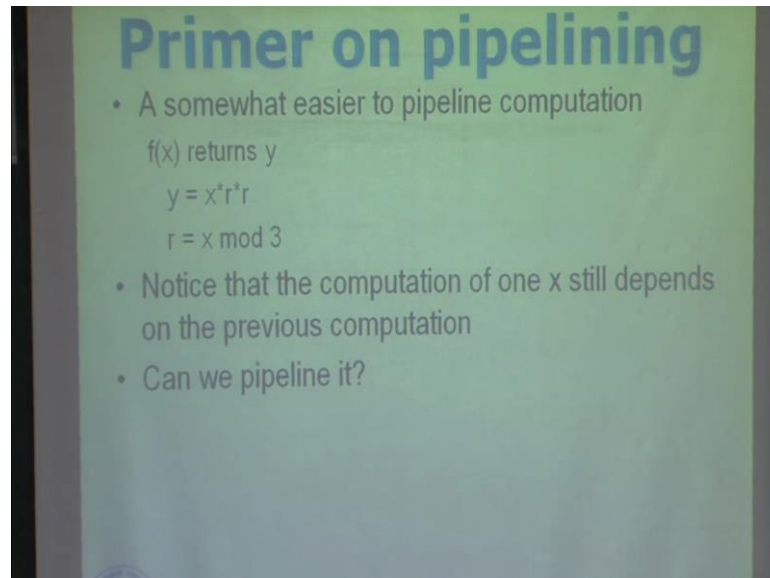
So, it should be fine are we good no one or not right, so here we have a bad edge crossing this line if you what is actually happening r is as usually not ready when you need it the data is not ready yet.

So, here is the hazard rule keep this in mind source stage of a data comes after the destination stage; that means, major problem in pipelines and there is no easy solution that can win back the lost performance in reverse. The only solution is we have to wait for once again, so in general you have to wait for a number of cycles that is equal to the distance between the source stage and the destination stage.

We discuss it the distance as 1 if you if you stall this particular stage for a month cycle you can get the right time the other one x times r square. So, here you can easily see the

problem right r has far reaching influence it goes to the first stage, actually we need from the very beginning of the computation, so latency is say 2 M plus mod 3. So, we are trying to pipeline it, there is a problem, so these are right, you can do it, so it is the same problem right here line, it is the same episode.

(Refer Slide Time: 45:33)



Primer on pipelining

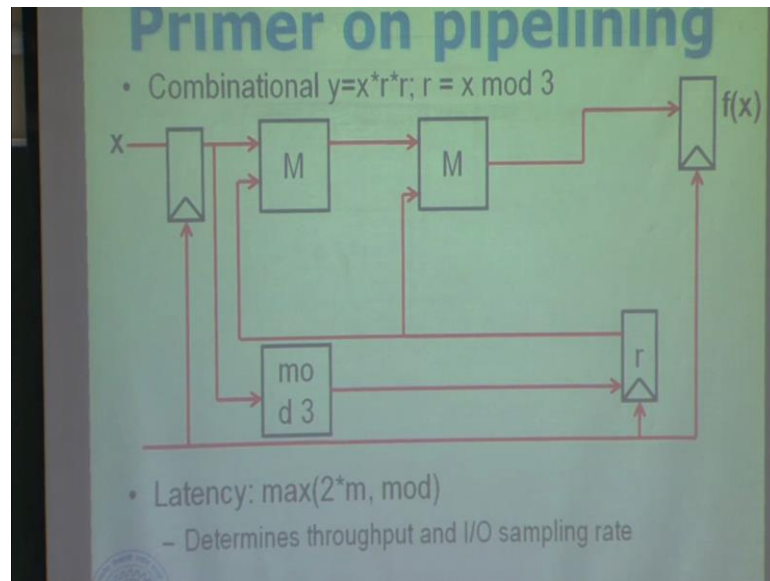
- A somewhat easier to pipeline computation

$f(x)$ returns y
 $y = x^2 r^2$
 $r = x \bmod 3$

- Notice that the computation of one x still depends on the previous computation
- Can we pipeline it?

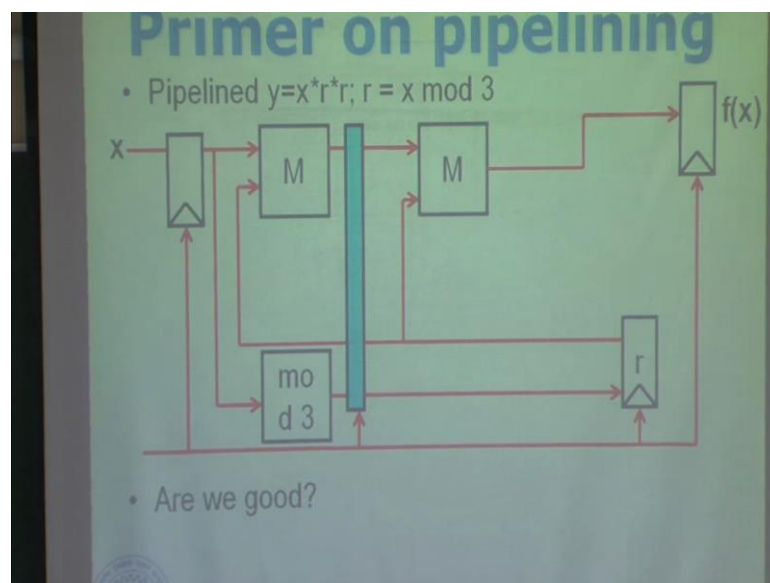
So, these are somewhat easier to pipeline computation, so we will keep x equal to x into r square and we will keep r to be equal to $x \bmod$ instead of $y \bmod$. So, it will come to r what, but remember that the computation of one x^3 depends on the previous computation because the r value comes from the previous x . Actually, it is not this r that comes in the computation, so there is still a dependence that lays on the previous computation, so can we pipeline it and I will say yes.

(Refer Slide Time: 46:06)



Let us see how it will, so this is my combinational circuit and I bring the mod down here right I take x through to this r and the latency I will change and now my becomes max of two M coma mod alright I can log this one with parallel with these two.

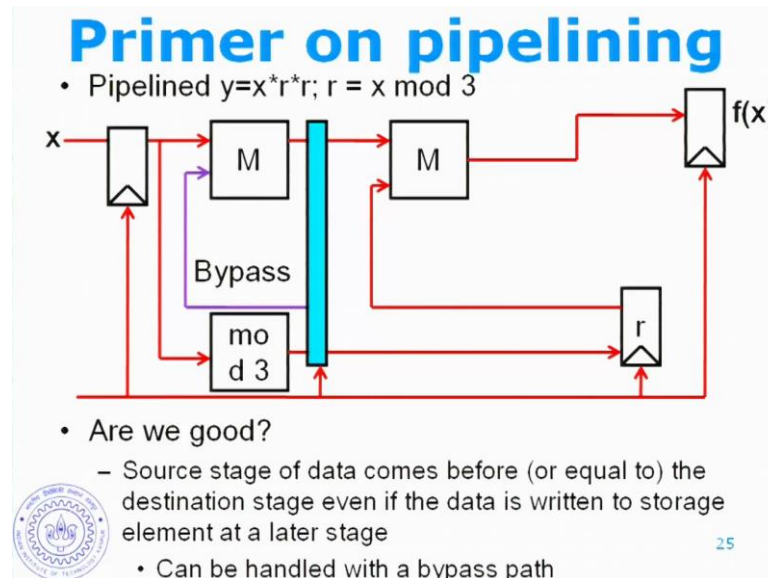
(Refer Slide Time: 46:35)



So, I will put a latch there are we good, how can you read it immediately, what kind of r

are am I using I can logically drag it here that is what he suggested. So, if you look at it, the latch also crosses a forward edge here, so that means, what I can do is when this pipeline finishes I can latch the value of r, so that the next step I can take the value of r from here alright that is called a bypass path.

(Refer Slide Time: 47:24)



So, if the source stage of data comes before or equal to the destination stage even if the data is written to storage element to the later stage we are fine can be handled with the bypass path. So, in this case this is a stage that is producing a value that is little after a long time, but the value is here actually ready for it to consume. So, if the destination stage is equal to this or before this we are we can handle that without any problem, this is all the hazard with bypass.

So, I think I am going to stop here, we have covered pretty much all the cases of pipelining that you will see why we have done pipeline upon. So, believe me pipelining a processor is orders a magnitude more difficult than a pipelining in simple computations. So, we will see the problems before we go to that I will spend a few minutes in next lecture trying to explain how to simulate a pipeline, because as we discussed in the first lecture in a computed design cycle usually start with simulations.

Before we finalize the design before it goes to for the design, so you have to understand how to simulate a pipeline if you are seeing very severe that why should it be any difficult. So, we will talk about that for a few minutes the next class and then we will take up pipeline running projects.