**Introduction:**

Retiming is used to minimize the cycle time or area of synchronous circuits by changing the position of the registers [1]. Cycle time in any circuit is more than or equal to the maximum critical path delay of the combinational logic in synchronous circuits.

By placing the registers in proper location this critical path delay can be reduced. We can use this methodology to convert a single cycle datapath into pipelined datapath.

A pipelined dapapath has hazard conditions. A problem with direct retiming here is, it does not change the functionality of the datapath. So we should take care to include (specify) the hazard detection hardware before retiming.

**Including hazard detection and Forwarding in datapath:**

A datapath can be pipelined into a number of stages depending on the individual operations and the time they take to perform them. Following is the general procedure for including the hazard unit for ‘n’ number of pipeline stages, ‘n’ being ‘5’ in the example I have taken here(for MIPS architecture).

Take a single cycle MIPS datapath.

Data memory

Reg file

Control unit

ALU

Instruction memory

Regfile

PC

wex-1

wex-2

wex

Memory operation

Read from register file

Execution

Fetch

Write back

In this datapath we can see that , it can be divided into 5 pipeline stages as there are 5 fundamental units that carry out different functions in the datapath.

If we assume a pipelined datapath with above individual blocks as different stages, we can see the possibility of the data hazards, when the destination of one instruction is going to be used in next few instructions. So these conditions are to be checked and proper hardware should be added to deal with the hazards.[2]

Conditions for hazards: When an earlier instruction in pipeline produces a result say in 3rd stage (execution stage) with a write enable and if the same data is required to be read in the next instructions which can be checked with a read enable, a hazard occurs based on the stage of pipeline the required data is in, before it is written into the proper register.

So in general for ‘n’ pipelined stages, if from 3rd stage write enable (we) signal is generated, assume it is wex for an instruction ‘x’ and wex-1 is the one produced by the previous instruction and so on, similarly read enable(re). By the time wex is generated, the older ‘we’ signal will be in the next pipelined stage and is called wex-1 .

BY using an algorithm (for example in a tool caller “T-piper” [2]), the hazard conditions hx-1 = rex and wex-1, hx-2 = rex and wex-2, hx-3 = rex and wex-3 , and so on are checked for when the destination register in one of the phases or instructions x-1, x-2, x-3,…. is equal to the source register in the ‘x’ phase.

In the MIPS we can analyze this condition manually fairly easily. If the destination register in phase x-1 is the same as the source register in stage x with wex-1 and rex being true, then a hazard hx-1 = rex and wex-1 occurs.

If the destination register in phase x-2 is the same as the source register in stage x where a read is required, with wex-2 and rex being true, then a hazard hx-2 = rex and wex-2 occurs.

If the destination register in phase x-3 is the same as the source register in stage x with wex-3 and rex being true, then a hazard hx-3 = rex and wex-3 occurs.

The final condition is stallpipeline= hx-1 or h x-2 or hx-3

So a hazard detection unit should be included to check these three conditions and stall the pipeline and should be placed in 2nd stage of pipeline.

But when the required data by an instruction ‘x’ is in the pipeline, although it hasn’t yet been written into proper register by an instruction x-1 or x-2 ….. Then bypassing the data helps in carrying out the correct operation without stalling the pipeline. This is called forwarding. The forwading conditions are to be checked and subtracted from the hazard conditions to avoid stalls when can be. But when forwarding is done care should be taken that the required data is not changed along the line before it is passed to proper location i.e.

The possible conditions of forwarding are, when f1 = hx-2(add $2,$3,$4; add $6,$2,$7) and

f2= hx-3(add $2,$3,$4; add $6,$4,$7; add $10,$2,$3) ,the destination and source registers of the ‘x-2’ or ‘x-3’ are assumed to be same as the source of ‘x’ here.

But here care should be taken that other recent hazards don’t occur i.e. hx-1 in the case of f1 and hx-1 and hx-2 in the case of “f2”.

So the forwarding conditions are modified as f1 = hx-2 and !hx-1, f2 = hx-3 and !hx-1 and !hx-2

These forwarding conditions are to be subtracted from the hazard conditions now

So stallpipeline.forward=stallpipeline & !f1 & !f2

A forwarding unit to check the above conditions is placed in the 3rd stage in MIPS, to forward the correct values in execution stage.

* Here one important point to notice is the addition of forwarding muxes which add to the delay in the 3rd stage of pipeline in MIPS.

This hardware is to be included in the datapath before retiming the single cycle datapath into pipelined.

Including these we get,

Memory operation

Read

HD.Unit

F.mux|Execution

F.Unit

Fetch

Write back

h

a

c

b

d

o

Retiming: Converting the above datapaht into retiming graph, we get

D1

D2

D3

D4

D5

1

0

0

0

0

Now, the delays of the individual units between the nodes are:

Between ‘h’ and ‘a’ : D1, path delay of instruction memory

‘a’ to ‘b’: D2, Max (path delay of Register file or control unit or hazard detection unit )

‘b’ to ‘c’: D3, path delay of forwarding mux + path delay of ALU

‘c’ to ‘d’: D4, path delay of data memory(memory+ address decoder)

‘d’ to ‘o’ : D5, path delay of register file.(writing back)

D={D1,D2,D3,D4,D5}

Path weight: is the number of registers between two nodes

In this graph, we have path weight of ‘1’ between ‘h’ and ‘a’, corresponding to ‘PC’. Rest of the path weights are zero.

W={1,0,0,0,0,0}

We now have to run a procedure called retime\_delay on the vectors D and W to obtain a retiming vector ‘R’ that minimizes the cycle time by inserting register at proper place.

**Retime\_delay[1] :** Assume a clock cycle time of ‘T’

Compute all the path weights and delays in the graph, and matrices W,D.

Sort the values of the elements of D.

Construct the inequalities:

 ri-rj <= wij for nodes i,j in the graph 1

 ri-rj <= W(vi,vj)-1 for all nodes vi  and vj forD( vi , vj)> T 2

Equation ‘1’ says that no path weights should be negative for legal retiming

Equation ‘2’ says that for a path between two nodes with critical path delay > T must have a pathweight of more than ‘0’.

Now, the problem is deciding on a cycle time T

‘T’ can be set initially to a value and above inequalities can be solved to check for legal retimting and if it is legal then reducing the ‘T’ to check for the legality with new ‘T’.

An efficient way of selecting ‘T’ is noticing that optimum ‘T’ matches path delay for some node pair.

So we can use a binary search on the entries of matrix D to find the optimum T.

So the formal procedure would be to solve the inequalities 1 and 2 using Bellman-Ford algorithm

For our problem of 5 stage pipeline D={D1,D2,D3,D4,D5}, ALU and memory operations take more time(D1,D3,D4), others take slightly less time(D2,D5).

**Here we retime manually** : The graph with included latency to have overall latency of ‘5’(number of pipeline stages) is

D1

D2

D3

D4

D5

1

0

0

0

0

4

To get one of the D’s as the cycle time ‘T’, we use binary search. But as all of them are almost equal we can easily see that moving registers to the right will decrease the critical path delay and finally at R={0,1,1,1,1}, we can see that the critical path delay is optimum.

**The sequence of retiming steps:**

D1

D2

D3

D4

D5

1

0

0

0

0

4

D1

D2

D3

D4

D5

1

0

0

0

4

0

R5=4

D1

D2

D3

D4

D5

1

0

0

3

1

0

R5=1

R4=3

D1

D2

D3

D4

D5

1

0

2

1

1

0

R5=1

R4=1

R3=2

D1

D2

D3

D4

D5

1

1

1

1

1

0

R5=1

R4=1

R3=1

R2=1

R1=0

The legal retiming vector to minimize the cycle time to obtain pipelined datapath from single cycle datapath is R={0,1,1,1,1}

**References:**

**1. G**. De Micheli, Synthesis and Optimization of Digital Circuits, New York: McGraw-Hill, 1994

**2.** Automatic Pipelining from Transactional Datapath Specifications, Eriko Nurvitadhi, James C. Hoe, Timothy Kam, Shih-Lien L. Lu.