DSD Final Project Presentation (MIPS)

Group member : 丁昱升、王珽
Professer : 吳安宇 教授
Date : 2019/06/20

Outline

- Baseline
  - Result
  - Special Design
  - Discussion about Cache
- Extension
  - Branch Prediction
    - Design
    - Analysis
  - Two-Level Cache
    - Design
    - Analysis
  - Multiplication & Division
    - Design
    - Analysis
- Work Distribution

Baseline - Result

- Area : 279585 (um^2)
- Synthesis cycle time : 2.3(ns)
- Simulation Cycle time : 2.3 (ns)
- Total simulation time : 5187.65 (ns)
- AT value : 1.4504 (10^9 * um^2 * ns)

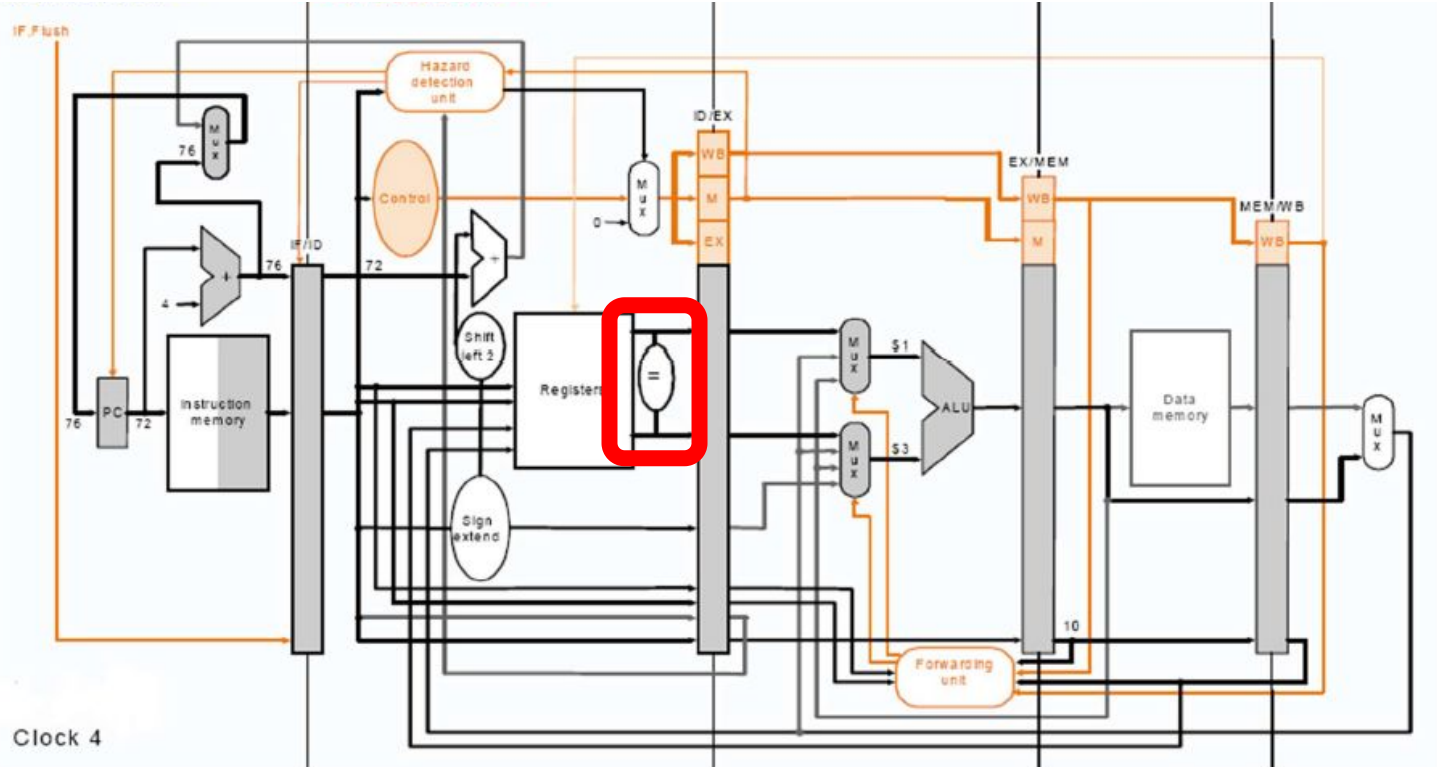
```
Total cell area:                279585.542478
Total area:                    2745744.823515
1
dc_shell>
```

```
ncsim> run
-----------------------------------------------

START!!! Simulation Start .....

-----------------------------------------------

FSDB Dumper for IUS, Release Verdi_N-2017.12, Linux, 11/12/2017
(C) 1996 - 2017 by Synopsys, Inc.
*Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file
*Verdi* : Create FSDB file 'Final.fsdb'
*Verdi* : Begin traversing the scope (Final_tb), layer (0).
*Verdi* : Enable +mda dumping.
*Verdi* : End of traversing.
*Verdi* : Begin traversing the scopes, layer (0).
*Verdi* : End of traversing.
------------------------- Simulation FINISH !!-------------------------
================================================================================

 \(^o^)/ CONGRATULATIONS!!  The simulation result is PASS!!!

================================================================================
Simulation complete via $finish(1) at time 5187650 PS + 0
./Final_tb.v:158              #(`CYCLE) $finish;
ncsim> exit
```

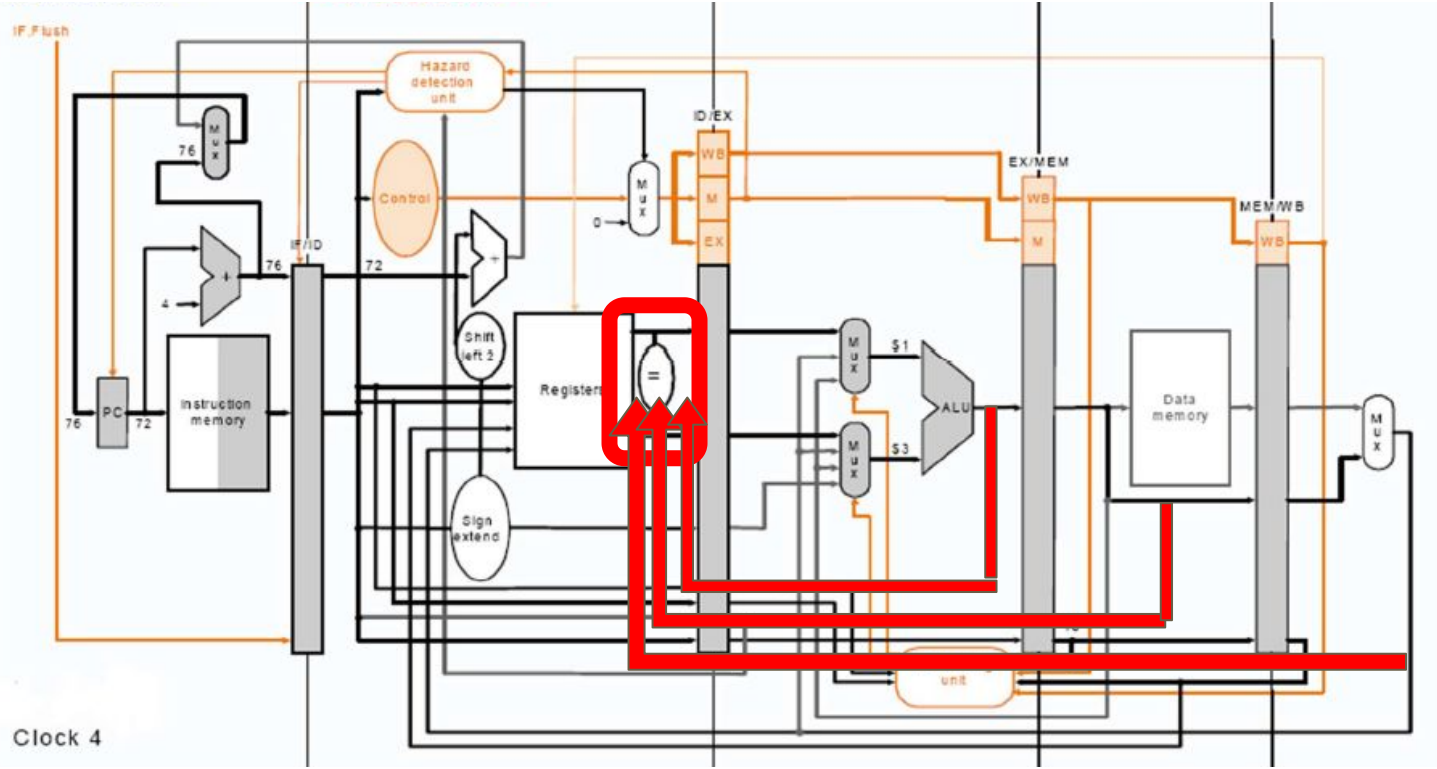
Baseline - Special Design

- Branch Forwarding Stall
- Why branch forwarding?
  - Check branch equality at ID stage

Baseline - Special Design

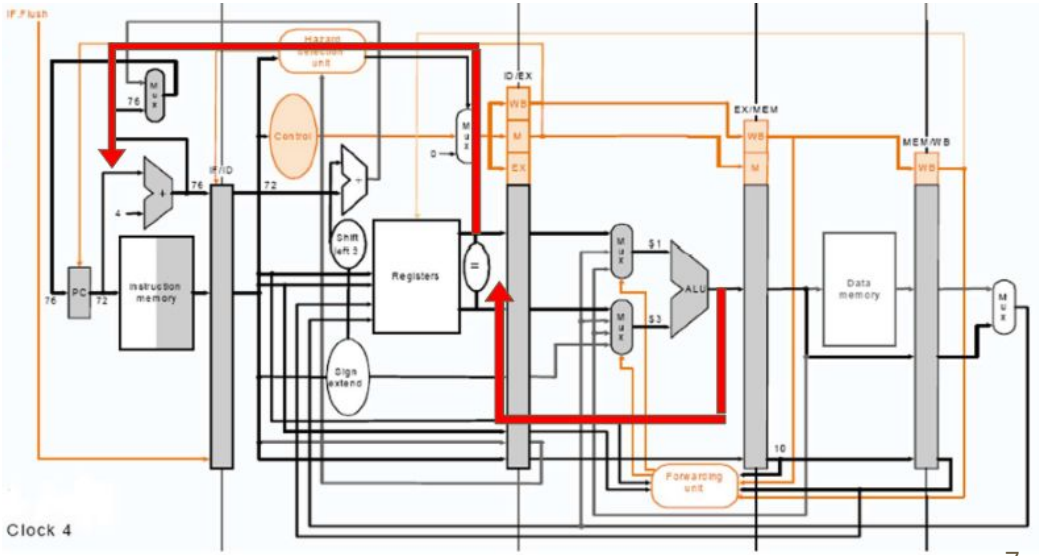


Baseline - Special Design

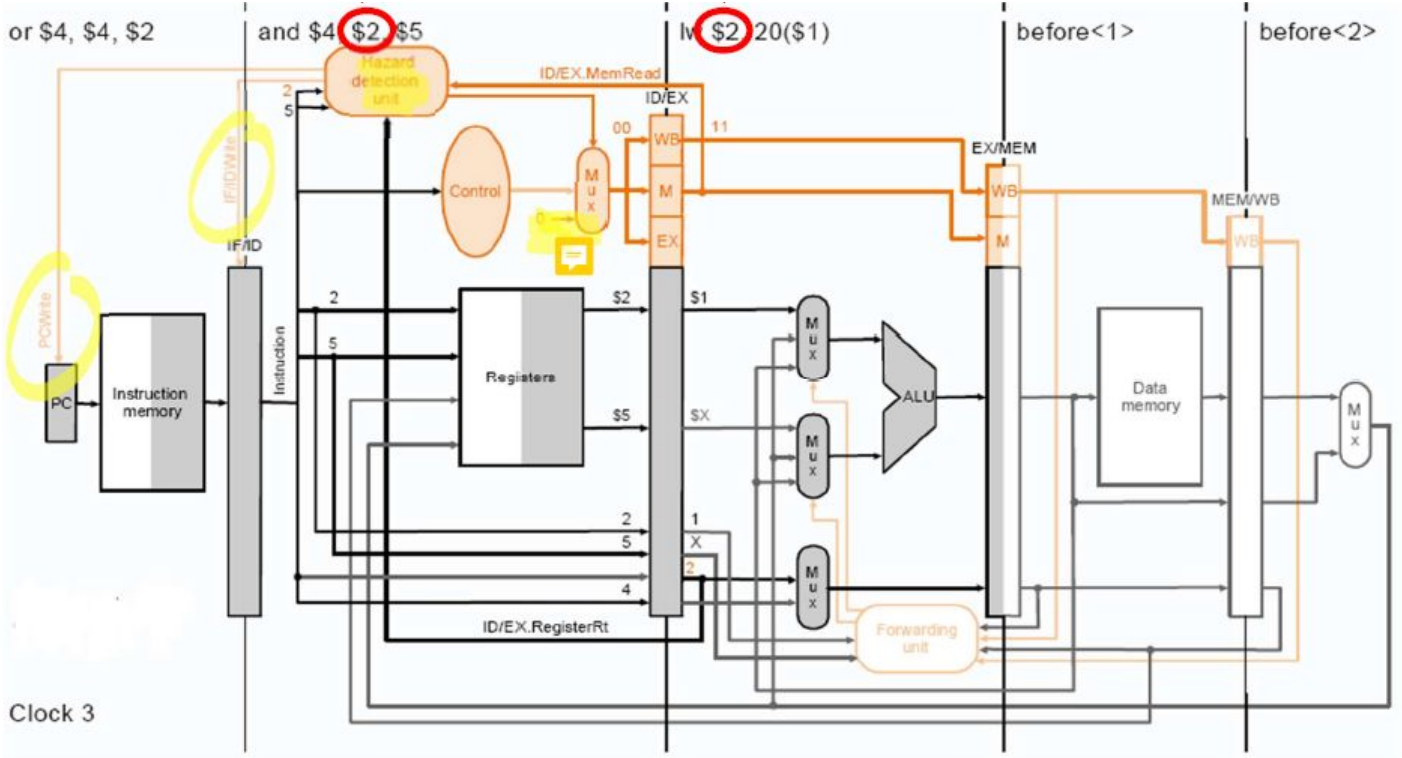


Baseline - Special Design

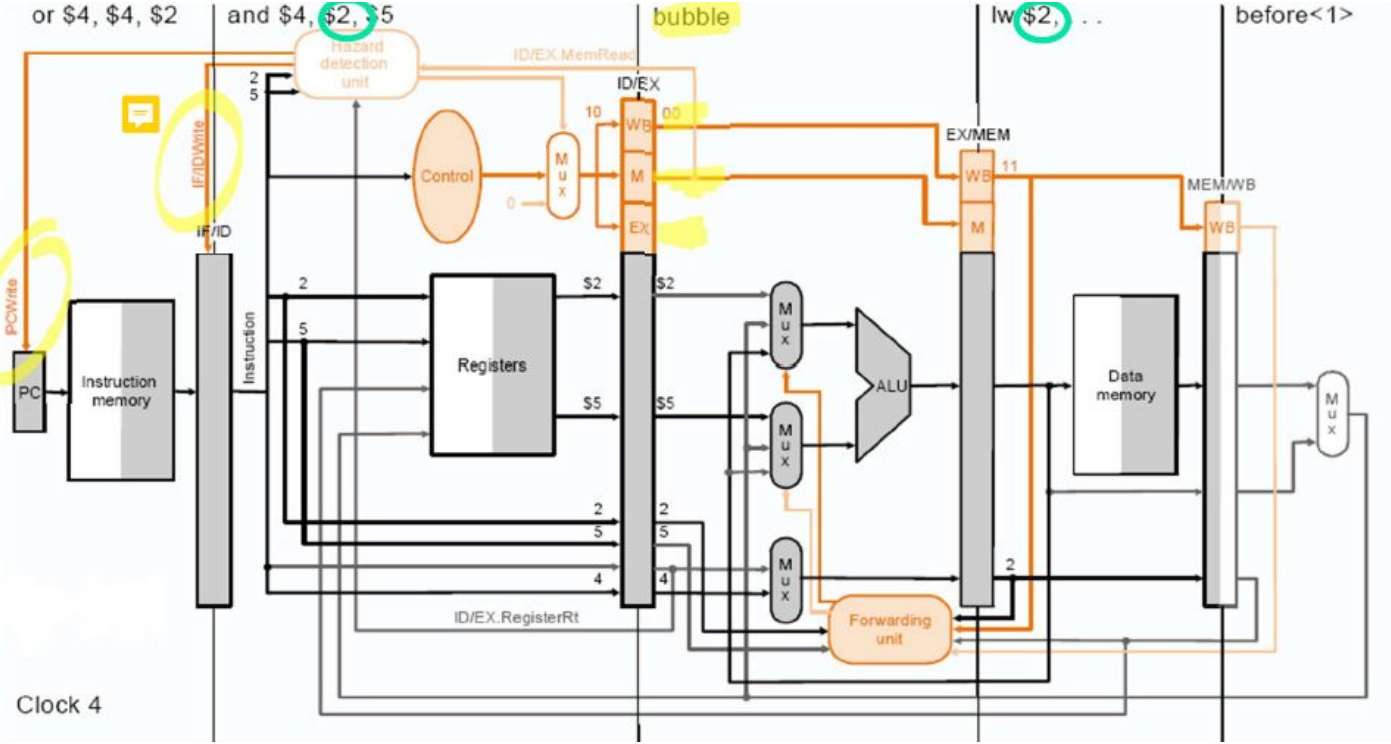
- What's the problem?
  - Critical path is too long
- How to fix the problem?
  - Don't forward through the path
  - "Stall" instead
  - Wait to forward from MEM
  - Can be combined with "lw" stall



Baseline - Special Design



Baseline - Special Design

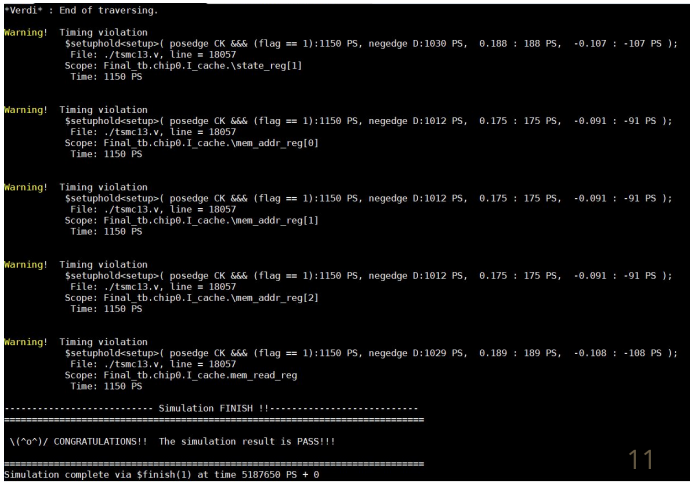


Baseline - Special Design

- The improvement
  - Before : cycle = 6.5
  - After : cycle = 4.31

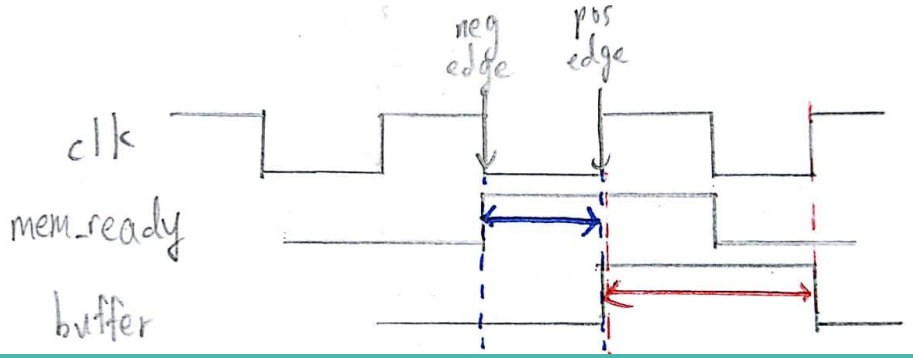
Baseline - Discussion about Cache

- Read-only I_cache
  - mem_wdata = 0
  - mem_write = 0
  - No need of dirty record
  - Reduction of number of states
- The improvement (?)
  - Before : area = 276837.450396
  - After : area = 279585.542478
  - Why?
    - The structure doesn't change a lot
    - Something happens in the process of synthesis?
  - Both cycle are 2.3
    - But read-only cache has no reset timing violation



Baseline - Discussion about Cache

- Buffer technique (Credit to 李承霖)
  - Data from memory change at negative clock edge
  - Cache design is positive edge triggered clock
  - If mem_ready is used as the flag of the change of next_data
    - It has to finish the computation in the half of the cycle!
  - How to fixed the problem?
    - Use a “buffer” as a flag
    - The buffer change its value at positive clock edge



Baseline - Discussion about Cache

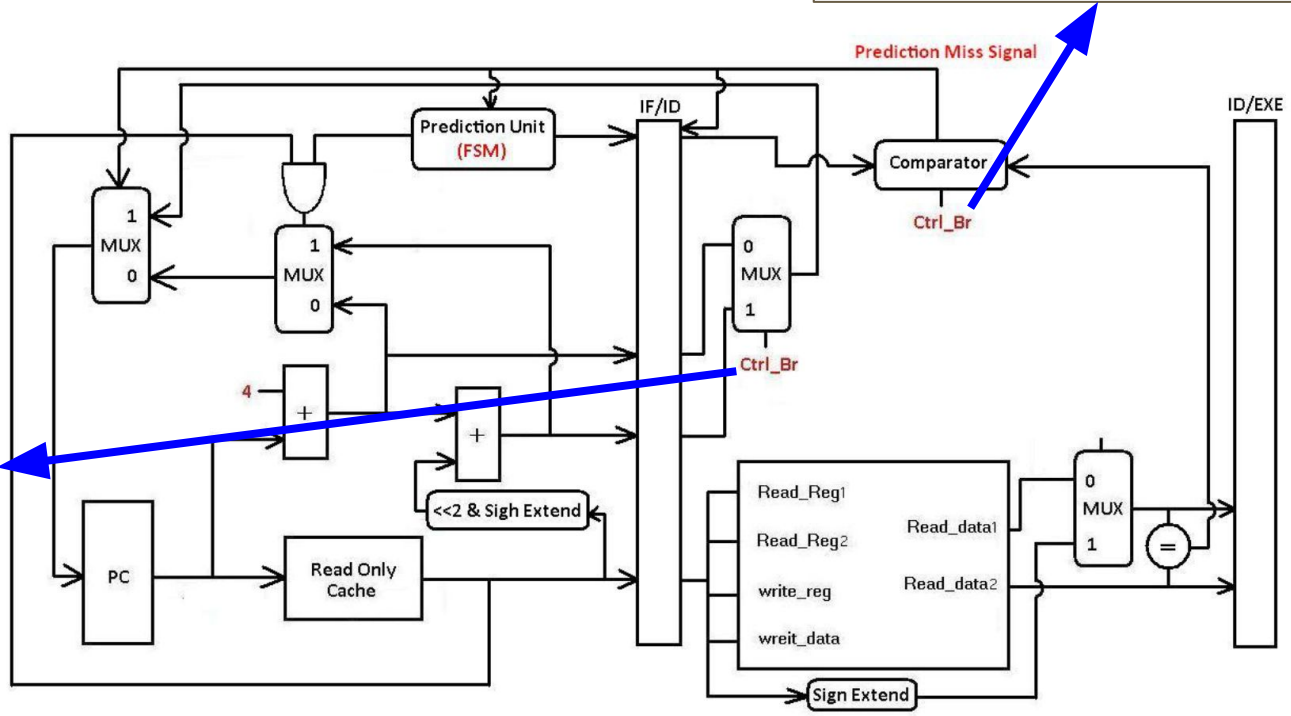
- The improvement
  - Before :
    - Cycle = 3.04
    - Area = 283533
  - After :
    - Cycle = 2.3
    - Area = 279585

Extension - Branch Prediction

- Design

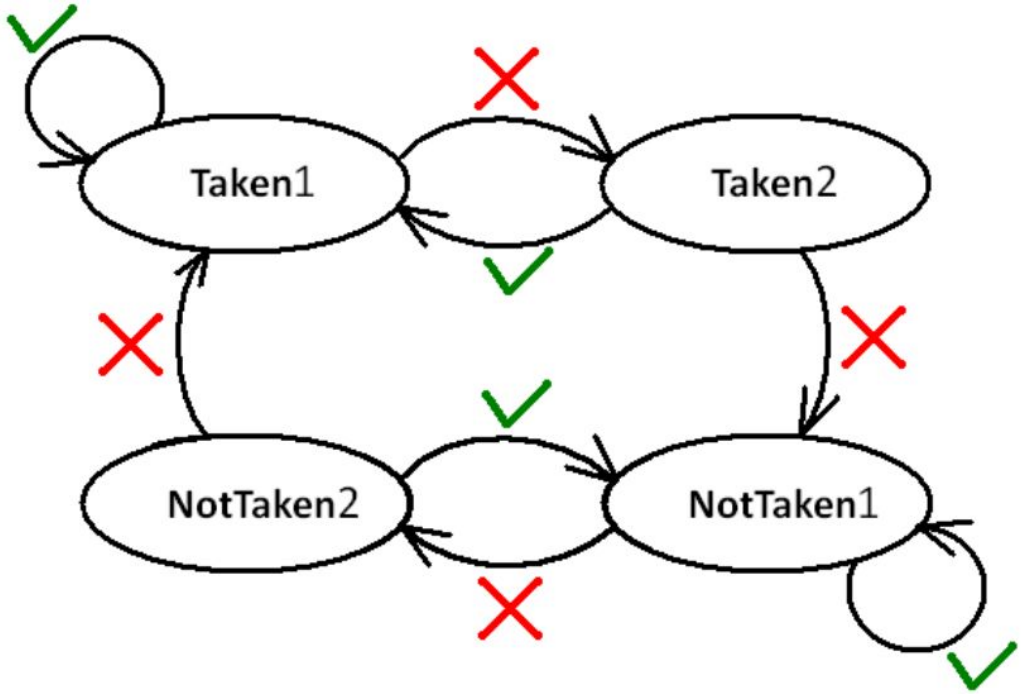
Control_Comparator: beq or bne

Control_MUX: if_branch



Extension - Branch Prediction

- Design



Take a look at TestBed

```
//0x2C// addi r12 r12 0x0001            // Part_B
//0x30// bne  r9  r12 0x0001 (to 0x38) // to Part_B_end
//0x34// j    0x0000010 (to 0x40)       // to Part_C
//0x38// beq  r8  r11 0xFFFC (to 0x2C) // Part_B_end, to Part_B
//0x3C// j    0x0000013 (to 0x4C)       // to Error
```

Supposed: interBranch (no branch/branch/no branch…)

Reality: two Branch in a row

Take a look at TestBed

Longer Time!!!!!

Not Branch

```
//0x1C// addi r11 r11 0x0001           // Part_A
//0x20// beq  r8  r11 0x0002 (to 0x2C) // to Part_B
//0x24// j    0x0000007 (to 0x1C)      // to Part_A
```

Branch

```
//0x40// addi r13 r13 0x0001           // Part_C
//0x44// bne  r10 r13 0xFFFE (to 0x40) // to Part_C
```

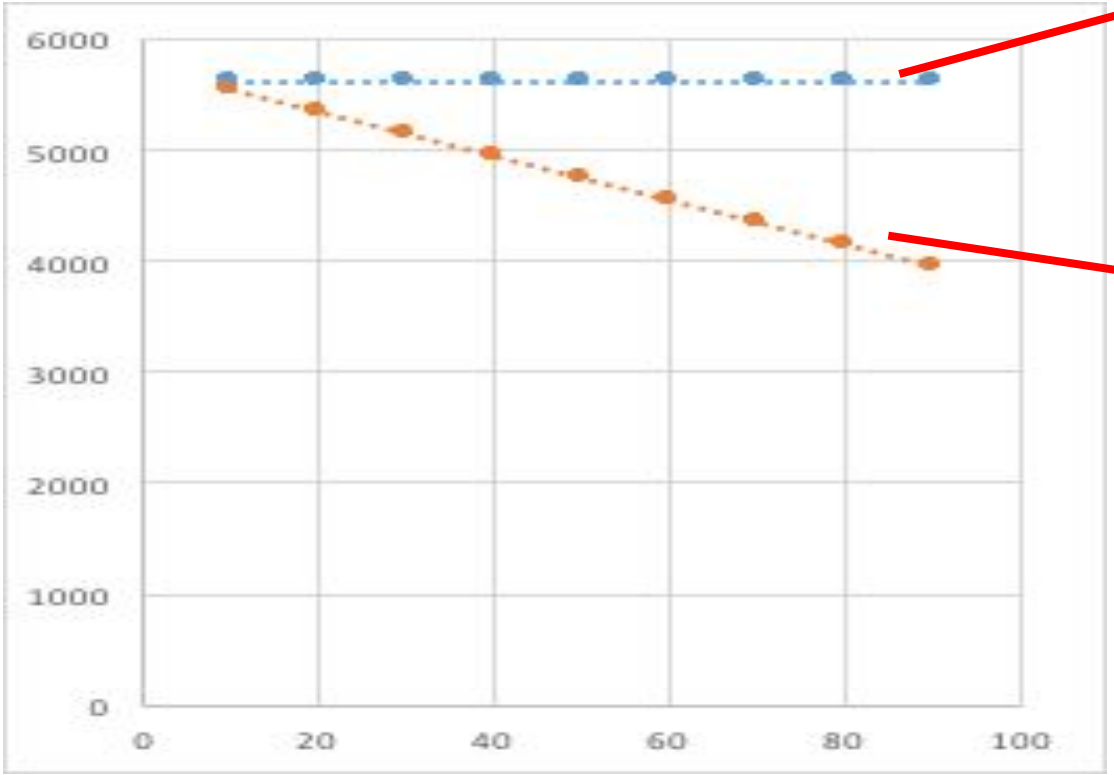
Extension - Branch Prediction

(notBr, Br)	CHIP_ini	CHIP_NEW
(90,10)	5615ns	5545ns
(80,20)	5615ns	5345ns
(70,30)	5615ns	5145ns
(60,40)	5615ns	4945ns
(50,50)	5615ns	4745ns
(40,60)	5615ns	4545ns
(30,70)	5615ns	4345ns
(20,80)	5615ns	4145ns
(10,90)	5615ns	3945ns

Initial: Always NotBranch

New: With Prediction Unit

Extension - Branch Prediction

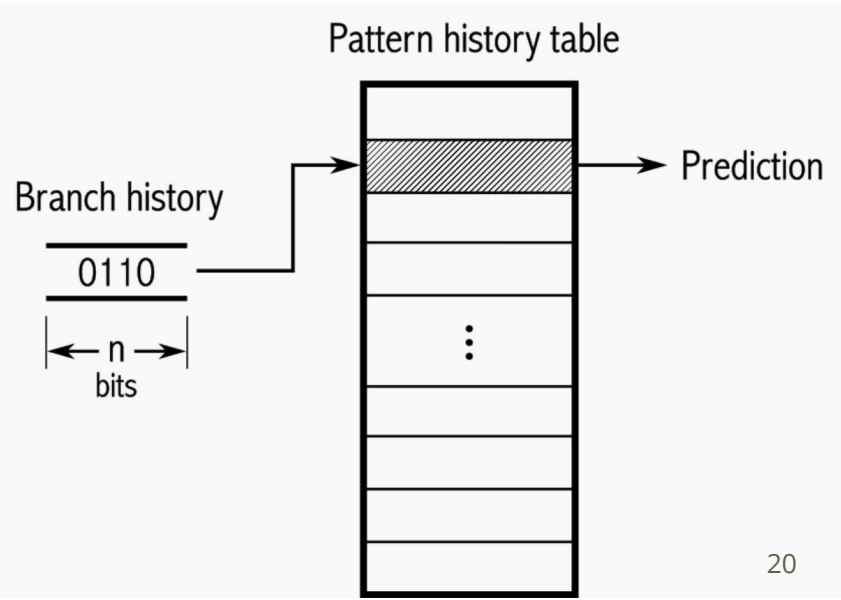


CHIP_ini

CHIP_NEW

Extension - Branch Prediction

- Original prediction unit is to simple.
- Use the Pattern History Table to improve.
- In our design, n = 2



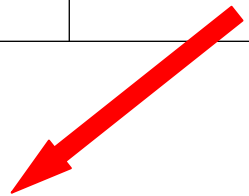
Extension - Branch Prediction

(notBr, Br)	CHIP_ini	CHIP_NEW	CHIP_HISTORY
(90,10)	5615ns	5545ns	5545ns
(80,20)	5615ns	5345ns	5345ns
(70,30)	5615ns	5145ns	5145ns
(60,40)	5615ns	4945ns	4945ns
(50,50)	5615ns	4745ns	4745ns
(40,60)	5615ns	4545ns	4545ns
(30,70)	5615ns	4345ns	4345ns
(20,80)	5615ns	4145ns	4145ns
(10,90)	5615ns	3945ns	3945ns

The same due to the TestBed

Extension - Branch Prediction

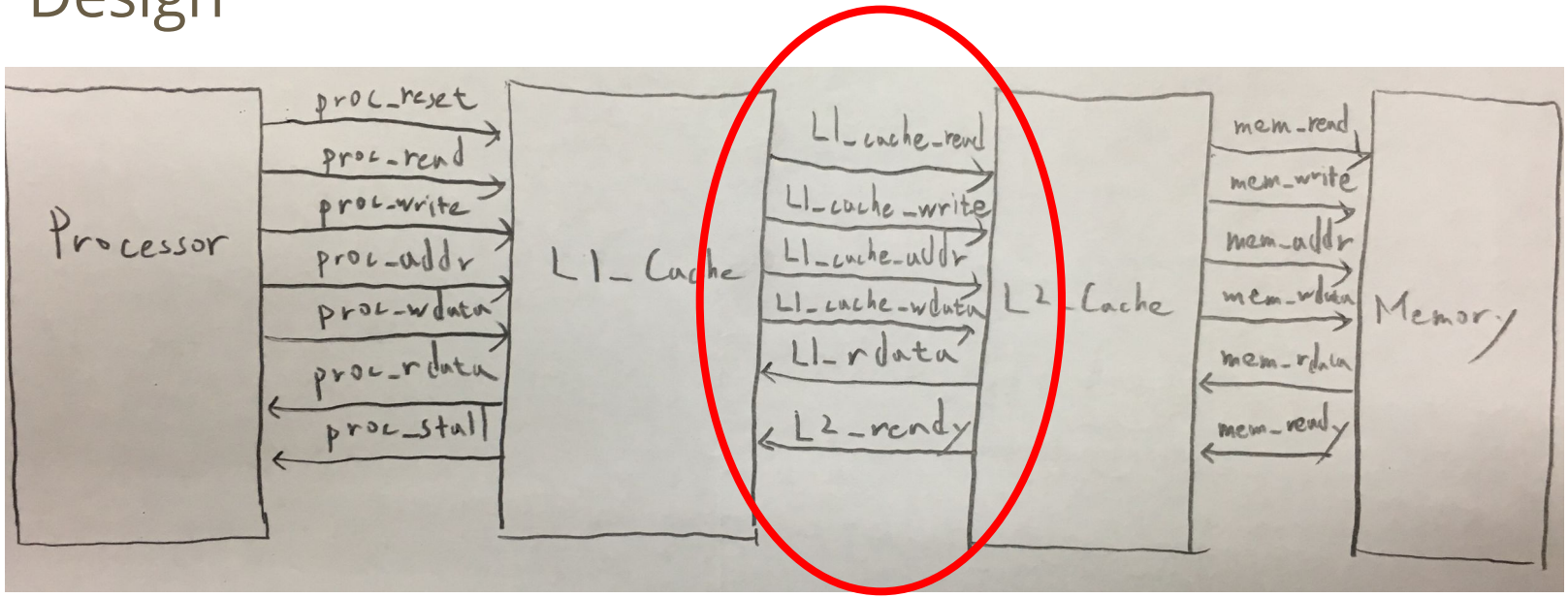
	CHIP_ini	CHIP_NEW	CHIP_HISTORY
hasHazard	21295ns	22225ns	21285ns



Chip_HISTORY outstands

Extension - Two-Level Cache

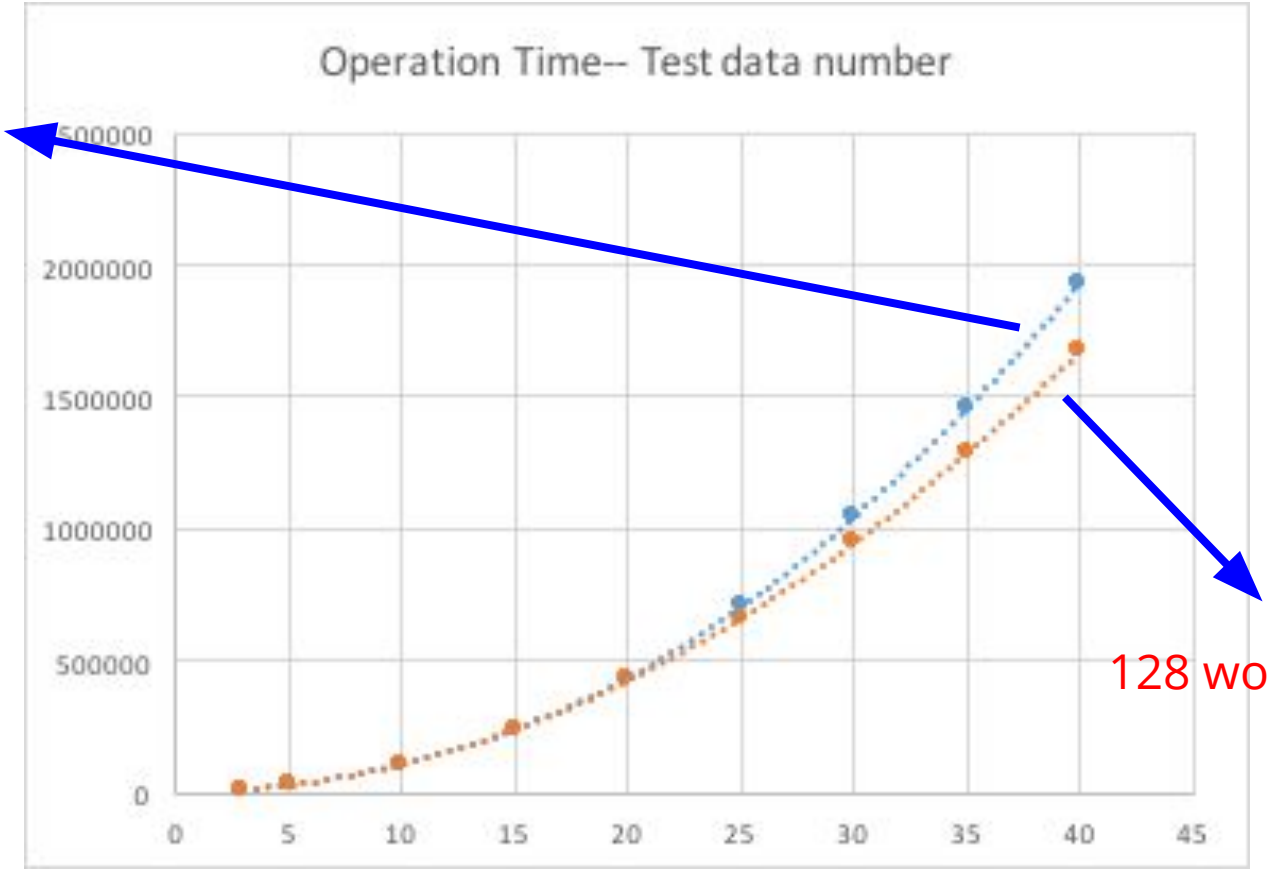
- Design



(nb, incr)	no L2	16 Blocks 64 words	32 Blocks 128 words	64 Blocks 256 words	128 Blocks 512 words
HasHazard	160965ns	150365ns	150365ns	150365ns	150365ns
(3,3)	13185ns	13345ns	13345ns	13345ns	13345ns
(5,3)	30425ns	30625ns	30625ns	30625ns	30625ns
(10,3)	113295ns	109535ns	109535ns	109535ns	109535ns
(15,3)	267385ns	241385ns	241385ns	241385ns	241385ns
(20,3)	481175ns	437935ns	426335ns	426335ns	426335ns
(25,3)	753965ns	706035ns	662285ns	662285ns	662285ns
(30,3)	1085515ns	1046635ns	948735ns	948735ns	948735ns
(35,3)	1476305ns	1458845ns	1294495ns	1286685ns	1286685ns
(40,3)	1926095ns	1931945ns	1711195ns	1675635ns	1675635ns

(nb, incr)	no L2	16 Blocks 64 words	32 Blocks 128 words	64 Blocks 256 words	128 Blocks 512 words
(3,3)	13185ns	13345ns	13345ns	13345ns	13345ns
(3,5)	25155ns	25355ns	25355ns	25355ns	25355ns
(3,10)	74895ns	74395ns	74395ns	74395ns	74395ns
(3,15)	162995ns	151405ns	151405ns	151405ns	151405ns
(3,20)	286365ns	259655ns	258325ns	258325ns	258325ns
(3,25)	441155ns	403335ns	393495ns	393495ns	393495ns
(3,30)	627795ns	58665ns	556375ns	556375ns	556375ns

No L2 Cache



128 word L2 Cache

(nb, incr)	no L2	16 Blocks 64 words	32 Blocks 128 words	64 Blocks 256 words	128 Blocks 512 words
HasHazard	160965ns	150365ns	150365ns	150365ns	150365ns
(3,3)	13185ns	13345ns	13345ns	13345ns	13345ns
(5,3)	30425ns			30625ns	30625ns
(10,3)	113295ns			109535ns	109535ns
(15,3)	267385ns			241385ns	241385ns
(20,3)	481175ns	437935ns	426335ns	426335ns	426335ns
(25,3)	753965ns	706035ns	662285ns	662285ns	662285ns
(30,3)	1085515ns	1046635ns	948735ns	948735ns	948735ns
(35,3)	1476305ns	1458845ns	1294495ns	1286685ns	1286685ns
(40,3)	1926095ns	1931945ns	1711195ns	1675635ns	1675635ns

The bigger, the better?

Extension - Two-Level Cache

- Bigger Size in reality takes longer time!
- Cost should be taken into account.

Extension - Multiplication & Division

- Area :
  - 296237 - 279585 = 16652 (um^2)
- Synthesis cycle time : 2.3(ns)
- Simulation Cycle time : 4.27 (ns)
- Total simulation time : 4654.49 (ns)

```
Total cell area:                296237.035458
Total area:                    2866463.488674
1
dc_shell>
```

```
------------------------- Simulation FINISH !!-------------------------
================================================================================

 \(^o^)/ CONGRATULATIONS!!  The simulation result is PASS!!!

================================================================================
Simulation complete via $finish(1) at time 4654490 PS + 0
./Final_tb.v:158              #(`CYCLE) $finish;
ncsim> exit
```

Extension - Multiplication & Division

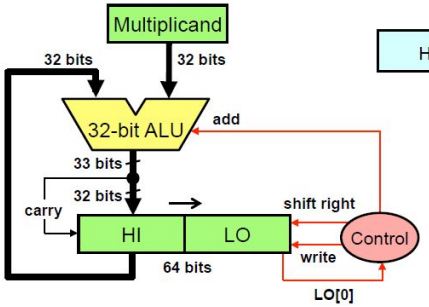
- Design (Iterative approach)

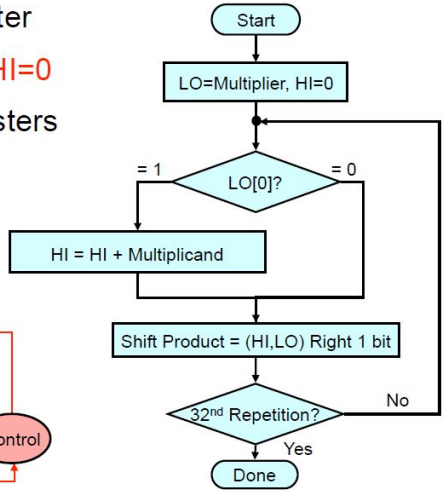
Multiplication

❖ Eliminate Multiplier Register

❖ Initialize LO = Multiplier, HI=0

❖ Product = HI and LO registers





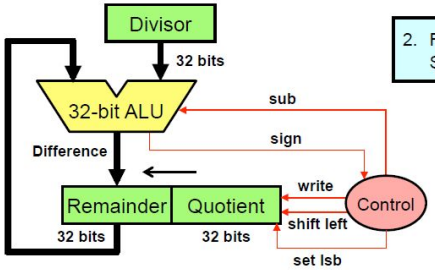
Division

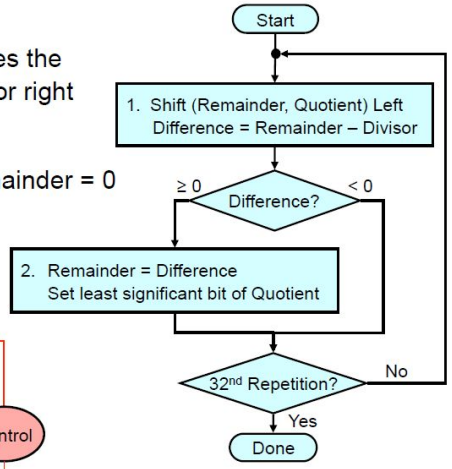
❖ Observation:

✧ Shifting remainder left does the same as shifting the divisor right

❖ Initialize:

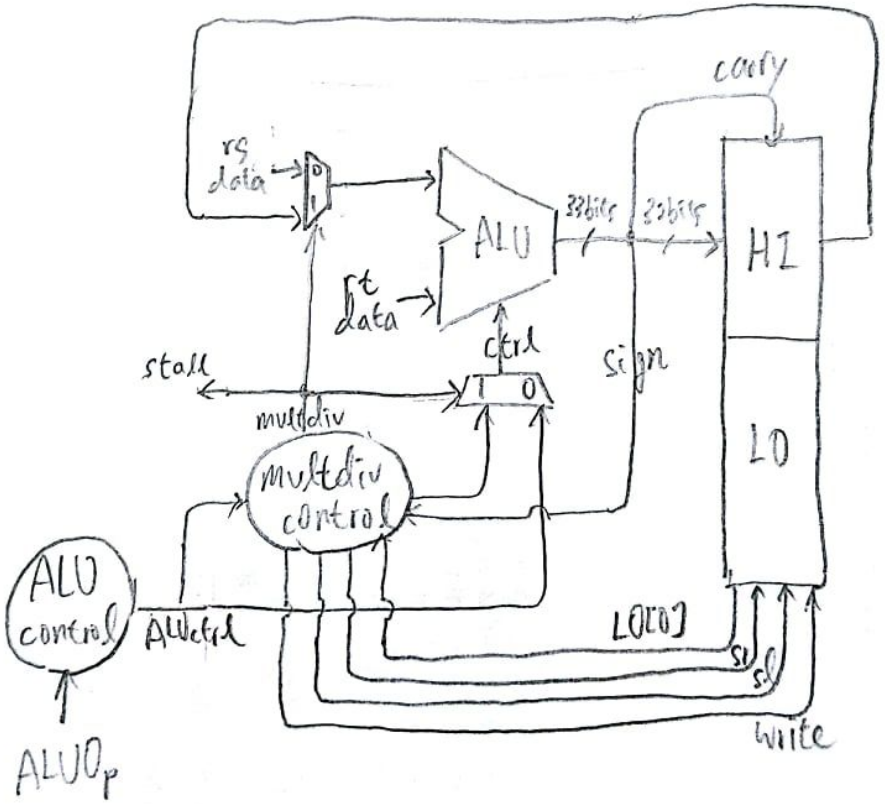
✧ Quotient = Dividend, Remainder = 0





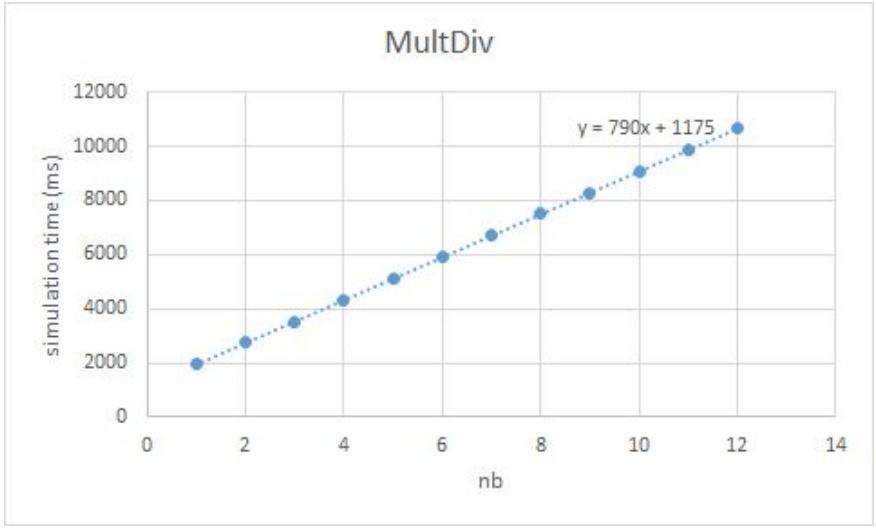
Extension - Multiplication & Division

- Design
  - Combined the two design
  - Share the ALU
    - By using mux
  - Stall when iteration



Extension - Multiplication & Division

- Analysis
  - Change nb
  - Linear change in simulation time



nb	sim_time
1	1965
2	2755
3	3545
4	4335
5	5125
6	5915
7	6705
8	7495
9	8285
10	9075
11	9865
12	10655

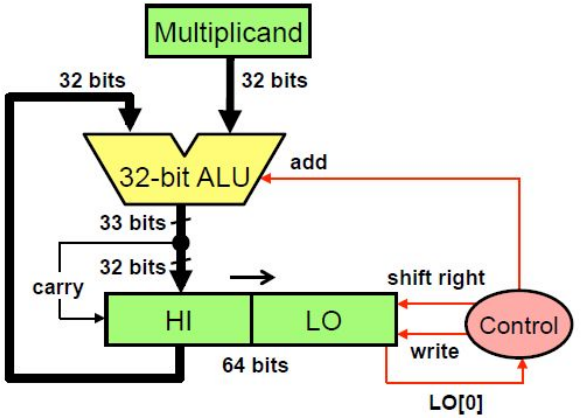
Extension - Multiplication & Division

- Every operation takes 32 cycles
  - May be a waste for small numbers
- Possible improvement
  - Use a comparator before ALU (at ID stage)
  - e.g.
    - If multiplier < 2^16 : shift 16 cycles only
  - Optimization

❖ Eliminate Multiplier Register

❖ Initialize LO = Multiplier, HI=0

❖ Product = HI and LO registers



Work Distribution

- Baseline - 丁昱升
- Extension
  - Branch Prediction - 王斑
  - Two-Level Cache - 王斑
  - Multiplication & Division - 丁昱升

THANKS FOR LISTENING !!