DSD Final Project
Final Report

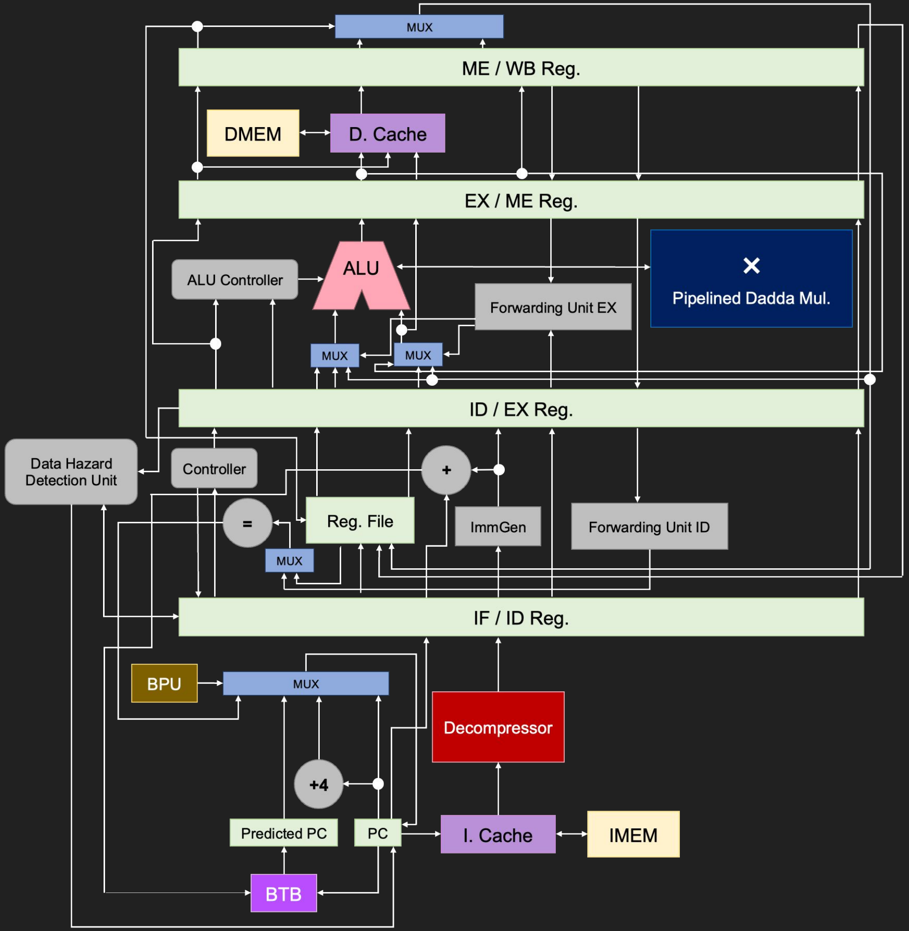
Hardware Implementation of Pipelined RISC-V

Group 7
B10901008 張禾牧
B10901016 邱巖盛
B10901078 歐信泓

Current Results

- Complete the whole architecture.
- Conducted some experiments (covered later).
- **Best Performance for now:**

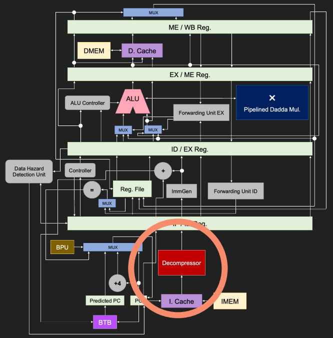
t_{cycle} (ns)	**2.82**
T_{BPred} (ns)	1287.330
T_{QSort} (ns)	361361.85
T_{Conv} (ns)	75656.37
A_{cell} (μm^2)	313764.3873
$A_{cell}T_{Conv}T_{QSort}$	8.58×10^{15}



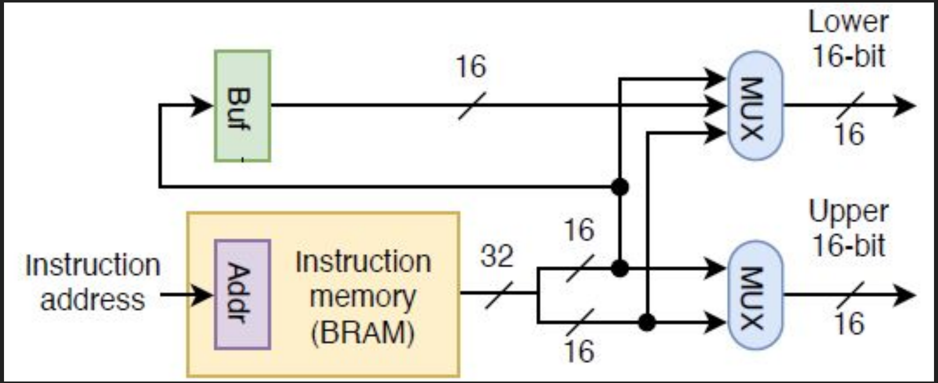
Extensions

Decompressor

- The instruction buffer will not always be valid (e.g. after jump).
- When an invalid buffer is required (`PC[1] && !inst_buffer_valid`), fetch lower instruction and stall for one cycle
- I cache addr. = `PC[31:2] + (PC[1] & inst_buffer_valid)`

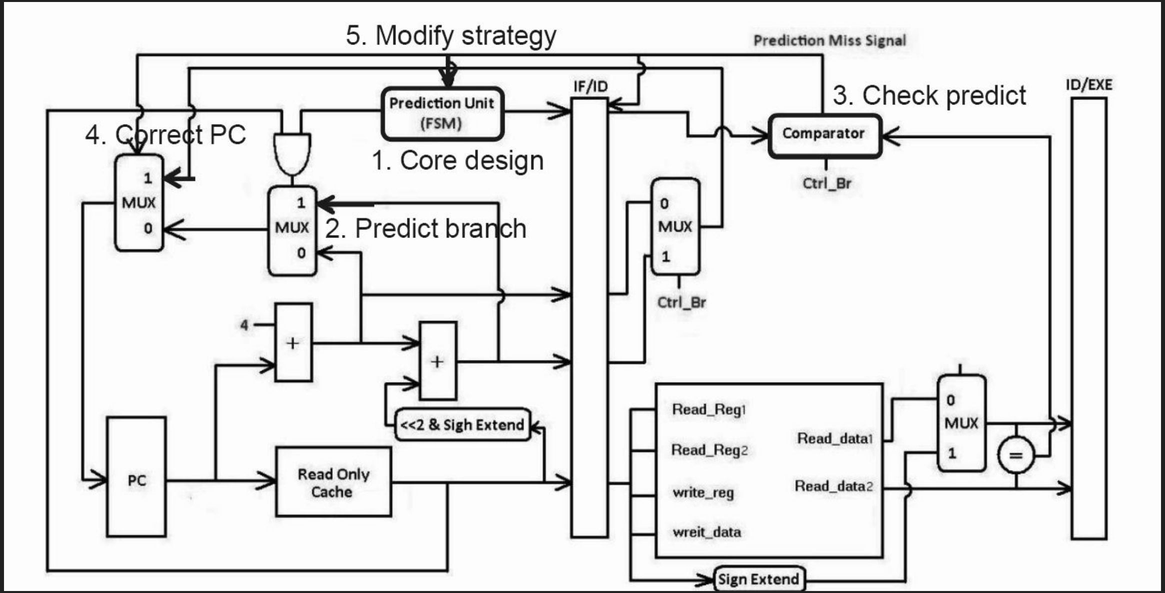


Decompressor



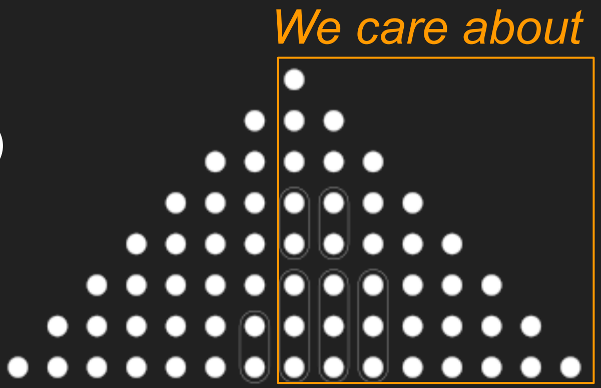
Branch Prediction

- Use two-bit saturation counter
- Save the `last_not_taken_PC` and use it if branch miss



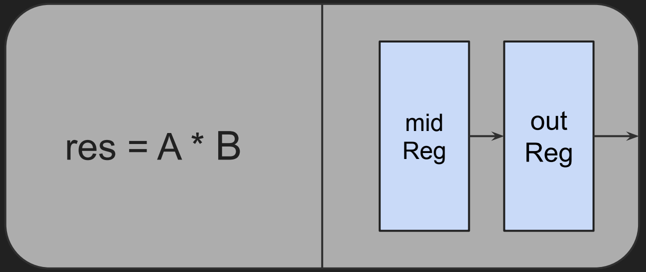
Multiplier

- Observations:
  - `mul` operation only requires lower 32 bits (~~`mulh`~~)
- Design
  - Verilog designware multiplier
  - Dadda multiplier with lower 32 bits



Pipelined Multiplier

- Use EX & MEM (& ID) for the pipelined multiplier
- Hazard: MEM_calculation (&& ID_calculation)
- Automatic Retiming
  - `set_dont_retime [get_registers …] false`
  - `set_optimize_register -design [get_designs mul_design]`



Dadda Multiplier

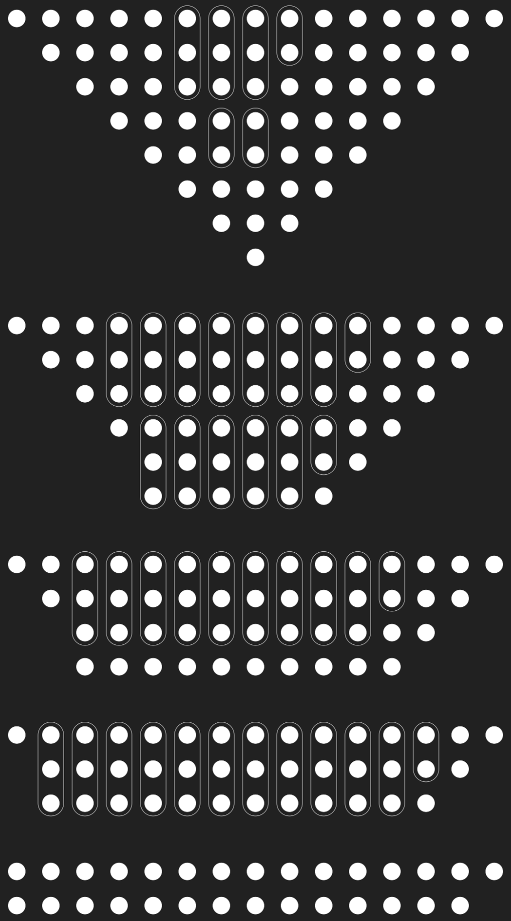
First we consider the classic 8-bit multipliers:

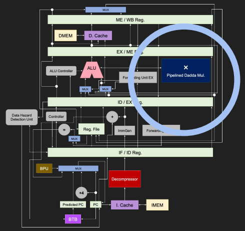
- *dadda_8*: basic 8-bit dadda multiplier
- *dadda_8_lower*: last 8 bits of the product

Partial Product Compression Priority:

1. Data from older stage > Data from newer stage
2. Temp. carry data > Temp. sum data

(critical path: and / or < xor)

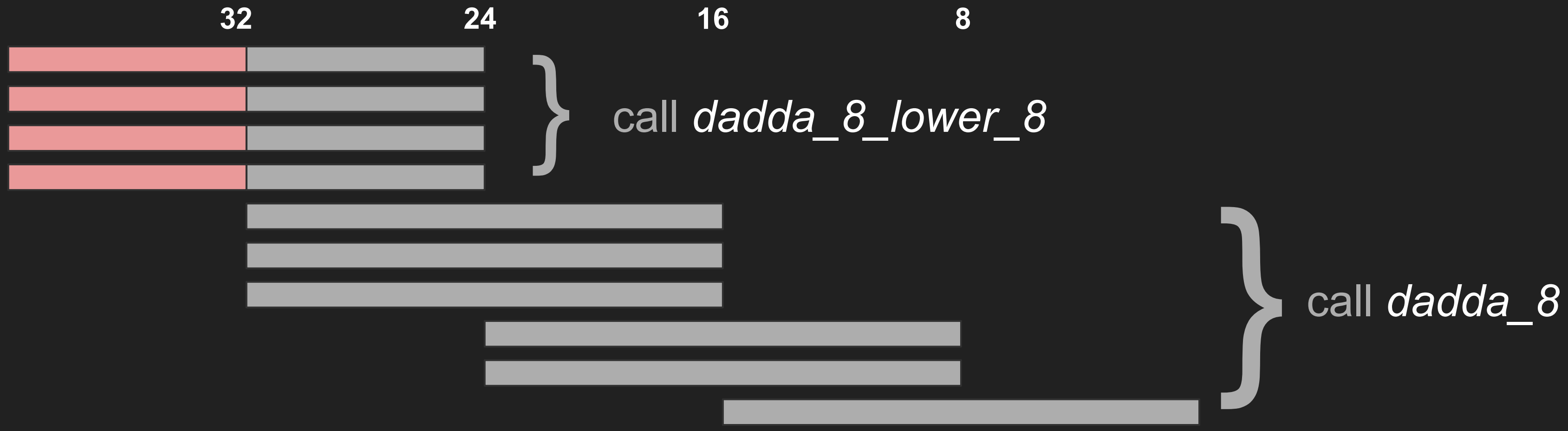




Dadda Multiplier (cont'd)

- Design a traditional 32-bit Dadda Multiplier is an extremely tough work.
- We can seperate 32-bit a, b into concatenation of 8-bit $\{a_0, a_1, a_2, a_3\}$ and $\{b_0, b_1, b_2, b_3\}$, thus

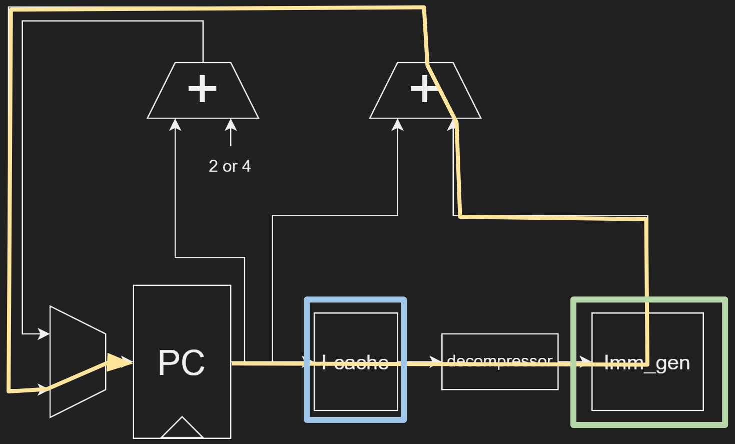
$$ab \bmod 2^{32} = \left(a_3 2^{24} + a_2 2^{16} + a_1 2^8 + a_0\right)\left(b_3 2^{24} + b_2 2^{16} + b_1 2^8 + b_0\right) \bmod 2^{32} =$$
$$\left(a_3 b_0 + a_2 b_1 + a_1 b_2 + a_0 b_3\right) 2^{24} + \left(a_2 b_0 + a_1 b_1 + a_0 b_2\right) 2^{16} + \left(a_1 b_0 + a_0 b_1\right) 2^8 + a_0 b_0$$



Observations

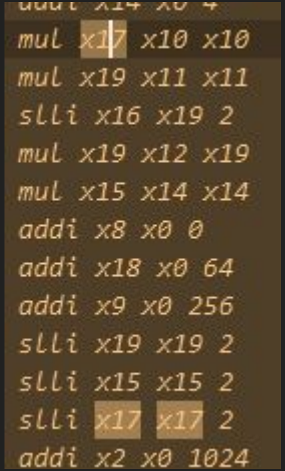
Issue about critical path

- Calculating the branch target consumes lots of time
- We use three different ways to reduce delay:
  - Instruction Cache optimization
  - Partial Immediate Generator
  - Branch Target Buffer

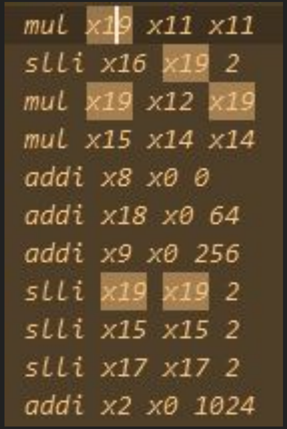


About the Convolution Testbench

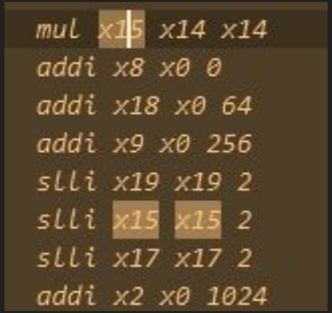
- Most `mul` results are not required immediately.
- More cycles can be used for `mul` calculations.



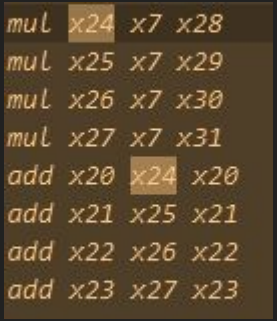
```
addi x14 x0 4
mul x17 x10 x10
mul x19 x11 x11
slli x16 x19 2
mul x19 x12 x19
mul x15 x14 x14
addi x8 x0 0
addi x18 x0 64
addi x9 x0 256
slli x19 x19 2
slli x15 x15 2
slli x17 x17 2
addi x2 x0 1024
```



```
mul x19 x11 x11
slli x16 x19 2
mul x19 x12 x19
mul x15 x14 x14
addi x8 x0 0
addi x18 x0 64
addi x9 x0 256
slli x19 x19 2
slli x15 x15 2
slli x17 x17 2
addi x2 x0 1024
```



```
mul x15 x14 x14
addi x8 x0 0
addi x18 x0 64
addi x9 x0 256
slli x19 x19 2
slli x15 x15 2
slli x17 x17 2
addi x2 x0 1024
```

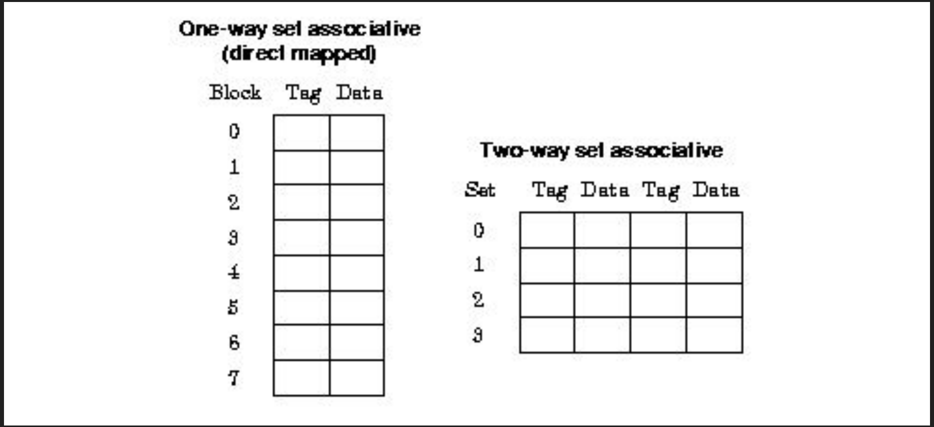


```
mul x24 x7 x28
mul x25 x7 x29
mul x26 x7 x30
mul x27 x7 x31
add x20 x24 x20
add x21 x25 x21
add x22 x26 x22
add x23 x27 x23
```

Optimizations

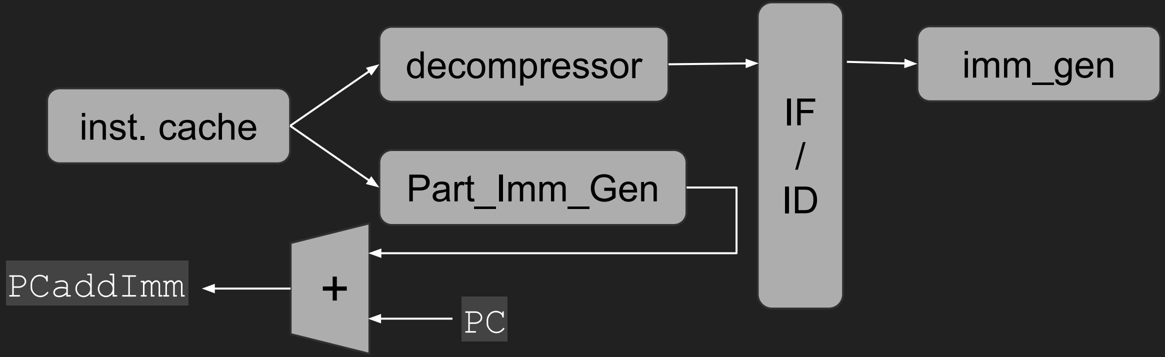
Cache Optimization

- *Direct-mapping* helps reduce critical path than *2-way* (about **0.3 ns**).
- Remove write-related function from I. Cache
  - area reduced (about 15000μm^2)



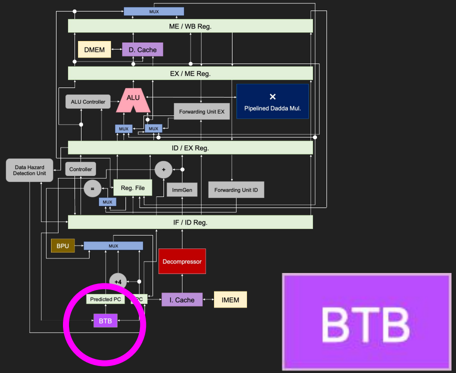
Partial Immediate Generator

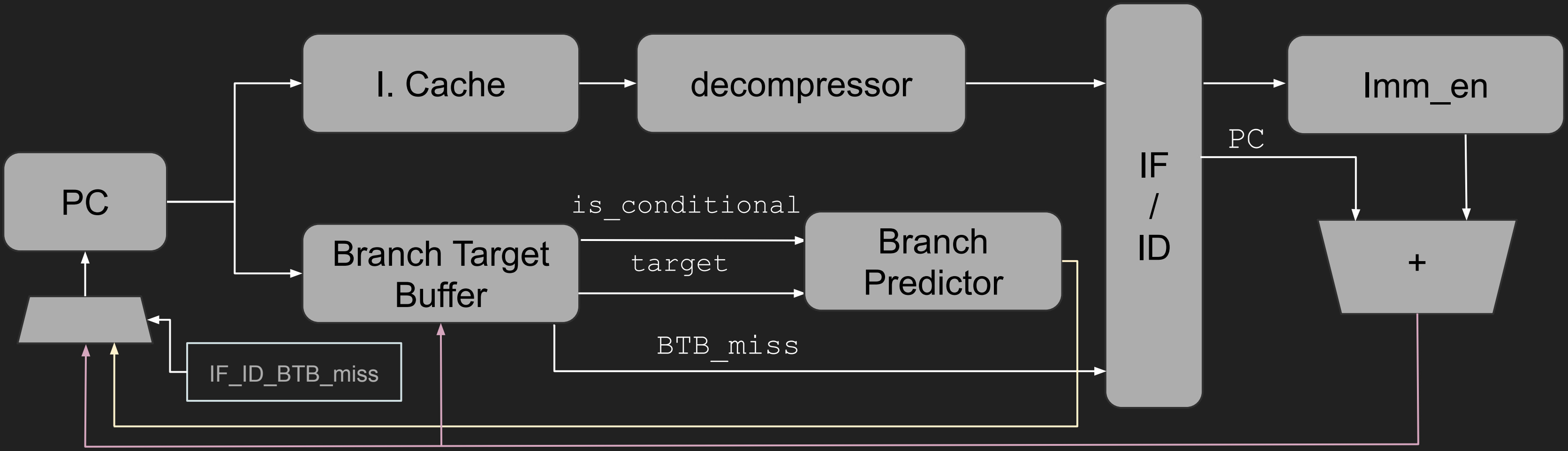
- A lot of time is wasted on calculating immediate because of the *decompressor*. (decompressor → imm_gen)
- Another unit *Part_Imm_Gen* for building a faster path for branch target
  - Use "decode" technique
  - Only provide correct immediate for (`C.`) `beq, bne, jal`
- Immediate is 21 bits (partial imm is 20 bits)



Branch Target Buffer

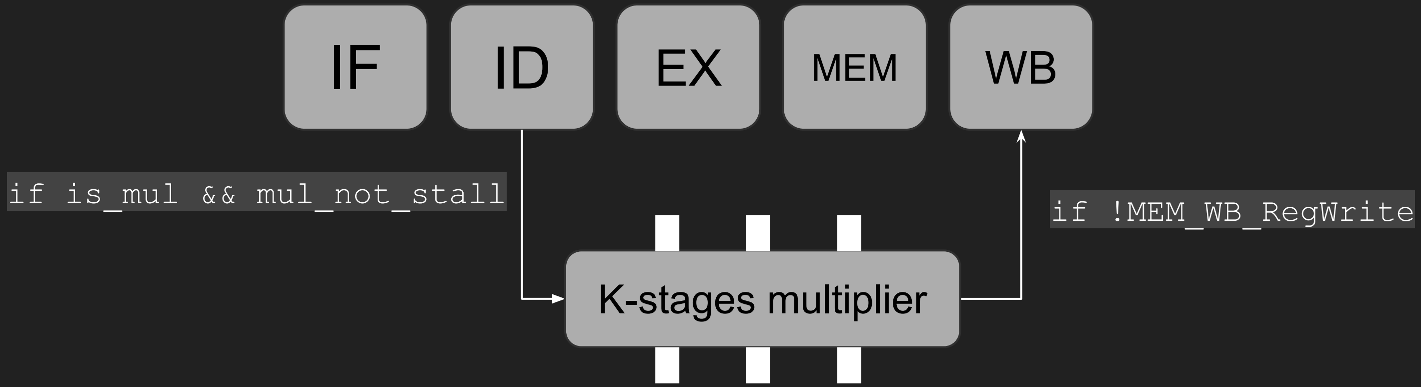
- Cache the target of branch instruction
- If there is any miss, the target will be calculated in ID stage & store into the buffer
- Cache Size: 2/4 block, *2-associative*





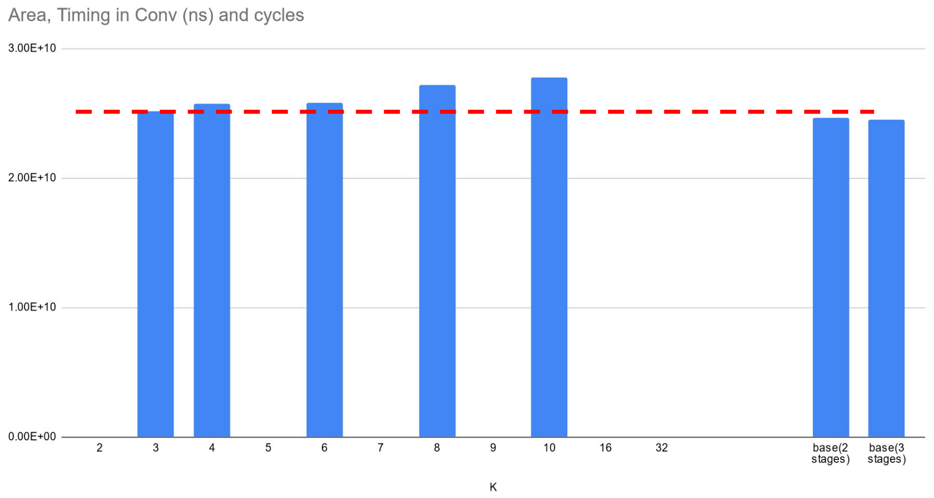
Out of order multiplication

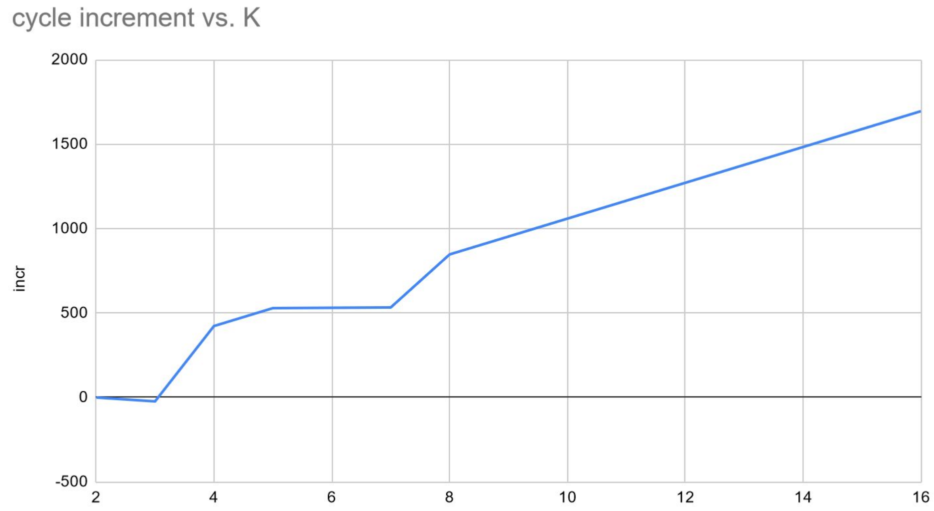
- It is for reducing the total area.
- No significant improvement (since the multiplier design doesn't changed).
- `mul` hazard: `[rs1 or rs2 ∈ any middle registers]`
- `mul_stall` = `mul_out_rd && MEM_WB_RegWrite`



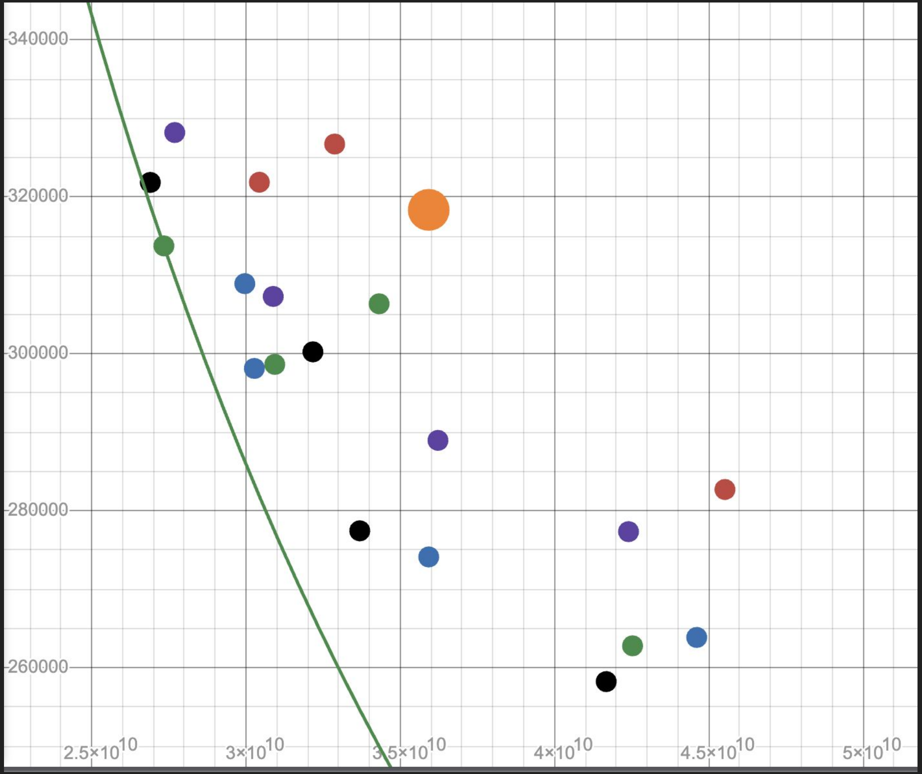
Result Analysis

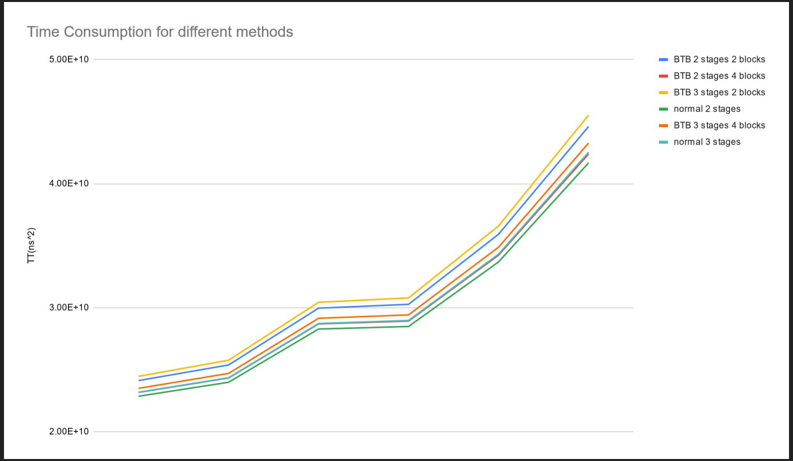
Out-Of-Order multiplier





Performance





Blue Dot:	BTB two stage(2 block)
Red Dot:	BTB three stage(2 blocks)
Black Dot:	normal two stage(2 block)
Green Dot:	normal three stage(2 blocks)
Purple Dot:	BTB two stage(4 blocks, 2 way)
Orange Dot:	Dadda multiplier

Green Curve: AT^2 curve passing the Green Dot

Work Assignment Chart

- **張禾牧**：

  Decompressor & Other Units, Optimization, Conduct Experiments

- **邱嚴盛**：

  Multiplier Connection, Optimization, Conduct Experiments

- **歐信泓**：

  Multiplier Connection, Synthesis Setup, Optimization, Conduct Experiments