EECS 470 Computer Architecture
Final project report

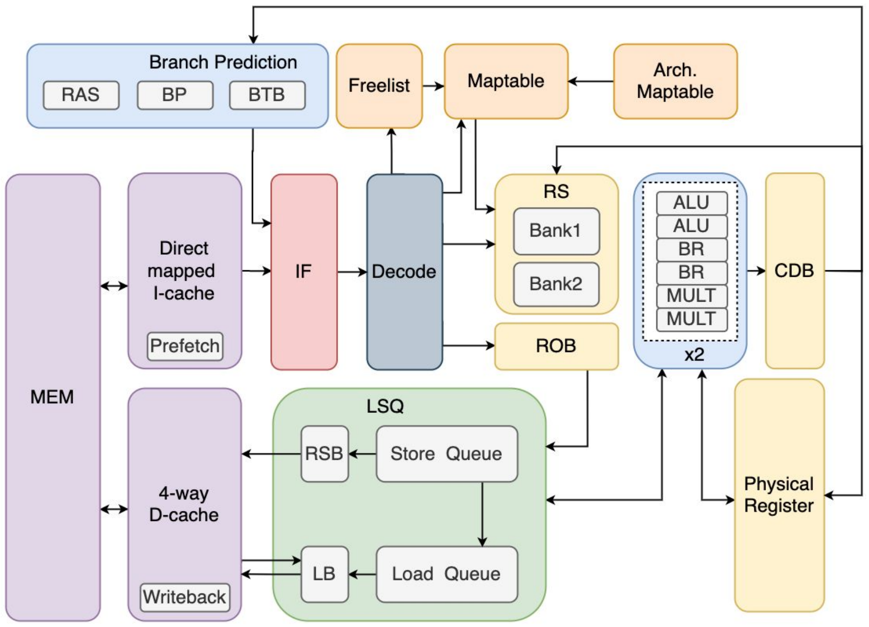
Group 6: Siliconfarmers
Jiajun Cao, Rui Sun, Wenbo Duan, Guoqin Yin
{jiajunc, rayss, wbduan, guoqin}@umich.edu

Introduction

In this project, we were required to design a modern processor with advanced features in synthesizable system verilog. We made our design choices based on our assessment of the difficulty and performance. In the end we did tests on our processor using different benchmarks and analyzed the performance.
We designed a R10K style 2-way superscalar Out of Order processor on RISC-V ISA. To achieve higher performance we added advanced features from four aspects, which are pipeline enhancement, branch predict enhancement, load-store processing enhancement and memory system enhancement. In the end our clock period used for synthesis is 12ns.

Design

- **Pipeline enhancement**
  The R10K style OoO processor pipeline we learned in class is a 7-stage pipeline including IF, ID, DP(dispatch), IS(issue), EX(ex), CM(complete) and RT(retire). In our design, we added RN(rename) stage between ID and DP stage to improve performance. For further improvement, we decided to build a 2-way superscalar system on this pipeline which can gain more performance. The architecture diagram of our system is shown below.



Our superscalar reservation station has 2 RS banks, each handles 1-way dispatch and 1-way issue. The size of RS bank is 8 entries so in total we have 16 entries in our superscalar RS. Our ROB has 32 entries and each entry will record PC, exception, halt, Tag_old, Tag_new, inst, rd/wr_mem, branch/predict_taken and other necessary information.
In our EX stage, we have 2 ALU, 2 8-stage MULT and 2 Branch resolution units. We can handle 2 branches at the same time and take the right result. Also the EX stage can send 2 load or store instructions to LSQ and the LSQ can solve 2 instructions at the same time. However due to the memory's bandwidth requirement, we can only send one load or store request to memory so we added two buffers between LSQ and Dcache.

- **branch prediction enhancement**
  Our branch prediction includes three components: direction predictor (DIRP), branch target buffer (BTB) and return address stack (RAS).
  1. DIRP: we designed a gshare direction predictor with global branch history table and per-address pattern history table. The size of PHT is 32 vertically and 32 horizontally, which means we use 5-bit of branch PC address to determine which pattern history table we use for each branch and 5-bit index to determine which pattern history state we use for predicting direction.
  2. BTB: The size of our branch target buffer is 32 as well. The structure of BTB is essentially a direct mapped cache, in which the tag is the branch PC address and the data is the corresponding target PC address. We use lower [6:2] bits of branch PC address to index BTB.
  3. RAS: In addition to the above features, we also added a return address stack(size is 16) and a predecoder. Predecoder is used to determine whether an instruction is a call PC instruction (JAL) or a return PC instruction (JALR). Every time when an instruction is pre-decoded as a JAL, it will be pushed into RAS or when an instruction is pre-decoded as a JALR, RAS will pop out the target return PC address.

- **load-store processing enhancement**
  To resolve load-store dependency, we added four components: store queue, load queue, load buffer, post-retirement store buffer.

1. Store queue: we use store queue to orderly record store instructions and add age logic to detect if store queue can forward value to a load instruction.
2. Load queue: load instructions are orderly recorded in the load queue. Also we add age logic in the load queue to detect if a load instruction executes early before store and cause memory violation. If it is , we set a memory violation bit for corresponding load instruction in ROB. When it's ready to retire, we roll back the whole system.
3. Load buffer: if load needs to get value from Dcache or memory, we push it into the load buffer and Dcache and memory can handle these read requests sequentially. Once Dcache returns the value, we store it in the load buffer therefore the common data bus(CDB) can read the load instruction from the load buffer for broadcast.
4. Post-retirement store buffer: Since dcache and memory need multiple cycles to write value. So after store instruction retires, we put it into the store buffer. So the cache and memory can handle those write requests sequentially.

- **memory system enhancement**

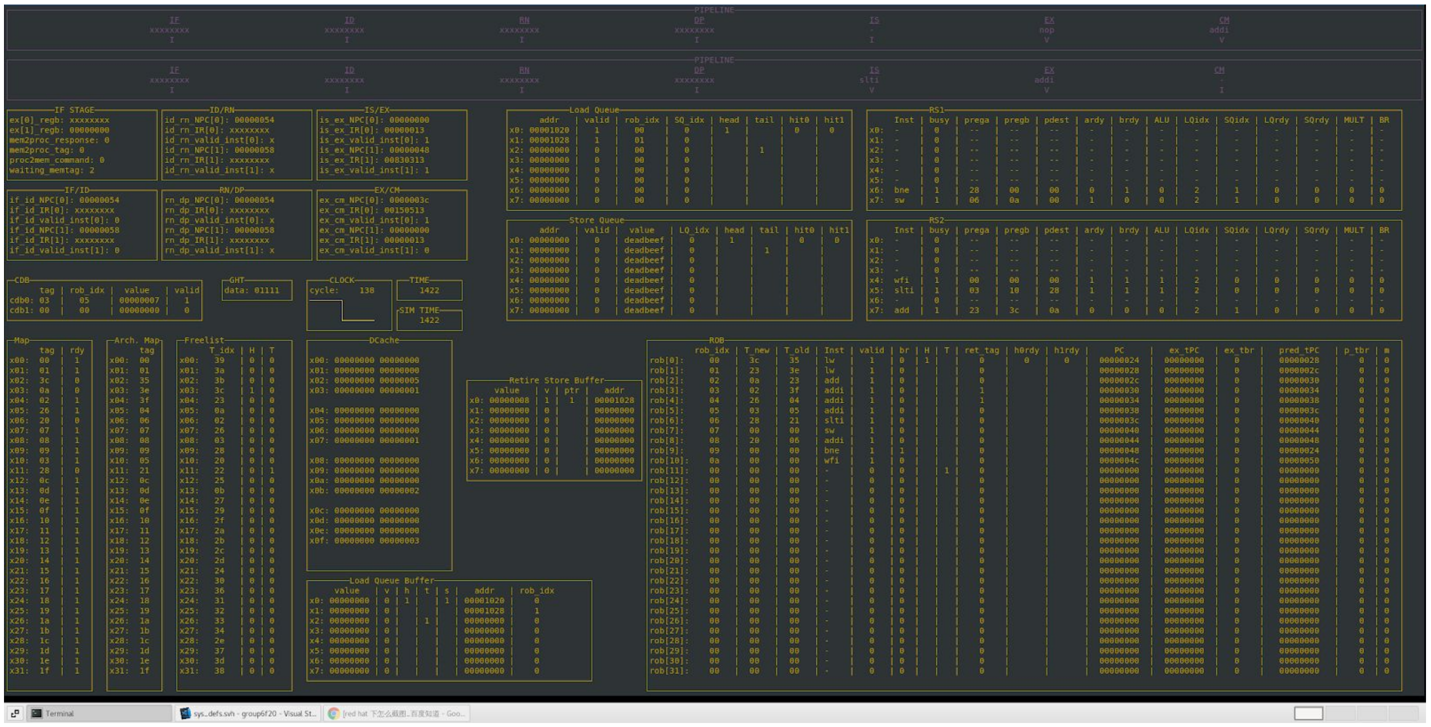
  In our system we designed Icache and Dcache separately.

  1. The Icache is a direct mapped cache with 16 cache lines. Each cache line has 8 Bytes. The prefetch policy in the Icache is that it will continuously prefetch instruction from the main memory until it fetches all zero data. There is a circular buffer that can record the response from memory and can send the correct instruction back to IF stage. If a branch or memory exception happens, the buffer will be reset and start again until a cache miss happens.
  2. For our Dcache, we designed a parameterizable N-way associative cache. Currently we are using a 4-way cache with 4 sets. This cache is a write-back cache and the eviction policy is LRU. We only set the recent bit when a write operation happens, which means if there is a write request from memory or there is a store hit from the processor, the recent bit of this cache line will be set. If a load instruction hits the cache, it will not change the recent bit. Also if there are load requests and store requests both happening from memory, the Dcache will handle the store first and start working on load requests until it resolves all store requests.

Debug and Performance Test

- **Visual Debugger**

  We implemented a visual debugger that can show the run-time information in each module.



```
PIPELINE
IF          ID          RN          DP          IS      EX      CM
xxxxxxxx    xxxxxxxx    xxxxxxxx    xxxxxxxx    -       nop     addi
I           I           I           I           I       V       V

PIPELINE
IF          ID          RN          DP          IS      EX      CM
xxxxxxxx    xxxxxxxx    xxxxxxxx    xxxxxxxx    slti    addi    -
I           I           I           I           V       V       I
```

```
IF STAGE
ex[0]_regb: xxxxxxxx
ex[1]_regb: 00000000
mem2proc_response: 0
mem2proc_tag: 0
proc2mem_command: 0
waiting_memtag: 2

ID/RN
id_rn_NPC[0]: 00000054
id_rn_IR[0]: xxxxxxxx
id_rn_valid_inst[0]: x
id_rn_NPC[1]: 00000058
id_rn_IR[1]: xxxxxxxx
id_rn_valid_inst[1]: x

IS/EX
is_ex_NPC[0]: 00000000
is_ex_IR[0]: 00000013
is_ex_valid_inst[0]: 1
is_ex_NPC[1]: 00000048
is_ex_IR[1]: 00830313
is_ex_valid_inst[1]: 1

IF/ID
if_id_NPC[0]: 00000054
if_id_IR[0]: xxxxxxxx
if_id_valid_inst[0]: 0
if_id_NPC[1]: 00000058
if_id_IR[1]: xxxxxxxx
if_id_valid_inst[1]: 0

RN/DP
rn_dp_NPC[0]: 00000054
rn_dp_IR[0]: xxxxxxxx
rn_dp_valid_inst[0]: x
rn_dp_NPC[1]: 00000058
rn_dp_IR[1]: xxxxxxxx
rn_dp_valid_inst[1]: x

EX/CM
ex_cm_NPC[0]: 0000003c
ex_cm_IR[0]: 00150513
ex_cm_valid_inst[0]: 1
ex_cm_NPC[1]: 00000000
ex_cm_IR[1]: 00000013
ex_cm_valid_inst[1]: 0
```

```
CDB
        tag | rob_idx |   value    | valid
cdb0: 03    |   05    | 00000007 |   1
cdb1: 00    |   00    | 00000000 |   0

GHT
data: 01111

CLOCK
cycle:    138

TIME
   1422

SIM TIME
   1422
```

```
Load Queue
      addr    | valid | rob_idx | SQ_idx | head | tail | hit0 | hit1
x0: 00001020 |   1   |   00    |   0    |  1   |      |  0   |  0
x1: 00001028 |   1   |   01    |   0    |      |      |      |
x2: 00000000 |   0   |   00    |   0    |      |  1   |      |
x3: 00000000 |   0   |   00    |   0    |      |      |      |
x4: 00000000 |   0   |   00    |   0    |      |      |      |
x5: 00000000 |   0   |   00    |   0    |      |      |      |
x6: 00000000 |   0   |   00    |   0    |      |      |      |
x7: 00000000 |   0   |   00    |   0    |      |      |      |

Store Queue
      addr    | valid |  value   | LQ_idx | head | tail | hit0 | hit1
x0: 00000000 |   0   | deadbeef |   0    |  1   |      |  0   |  0
x1: 00000000 |   0   | deadbeef |   0    |      |  1   |      |
x2: 00000000 |   0   | deadbeef |   0    |      |      |      |
x3: 00000000 |   0   | deadbeef |   0    |      |      |      |
x4: 00000000 |   0   | deadbeef |   0    |      |      |      |
x5: 00000000 |   0   | deadbeef |   0    |      |      |      |
x6: 00000000 |   0   | deadbeef |   0    |      |      |      |
x7: 00000000 |   0   | deadbeef |   0    |      |      |      |
```

```
RS1
     Inst | busy | prega | pregb | pdest | ardy | brdy | ALU | LQidx | SQidx | LQrdy | SQrdy | MULT | BR
x0:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x1:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x2:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x3:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x4:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x5:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x6:  bne  |  1   |  28   |  00   |  00   |  0   |  1   |  0  |   2   |   1   |   0   |   0   |  0   | 0
x7:  sw   |  1   |  06   |  0a   |  00   |  1   |  0   |  0  |   2   |   1   |   0   |   0   |  0   | 0

RS2
     Inst | busy | prega | pregb | pdest | ardy | brdy | ALU | LQidx | SQidx | LQrdy | SQrdy | MULT | BR
x0:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x1:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x2:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x3:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x4:  wfi  |  1   |  00   |  00   |  00   |  1   |  1   |  1  |   2   |   0   |   0   |   0   |  0   | 0
x5:  slti |  1   |  03   |  10   |  28   |  1   |  1   |  1  |   2   |   0   |   0   |   0   |  0   | 0
x6:  -    |  0   |  --   |  --   |  --   |  -   |  -   |  -  |   -   |   -   |   -   |   -   |  -   | -
x7:  add  |  1   |  23   |  3c   |  0a   |  0   |  0   |  0  |   2   |   1   |   0   |   0   |  0   | 0
```

```
Map                   Arch. Map      Freelist
      tag | rdy             tag            T_idx | H | T
x00:  00  |  1        x00:  00       x00:   39   | 0 | 0
x01:  01  |  1        x01:  01       x01:   3a   | 0 | 0
x02:  3c  |  0        x02:  35       x02:   3b   | 0 | 0
x03:  0a  |  0        x03:  3e       x03:   3c   | 1 | 0
x04:  02  |  1        x04:  3f       x04:   23   | 0 | 0
x05:  26  |  1        x05:  04       x05:   0a   | 0 | 0
x06:  20  |  0        x06:  06       x06:   02   | 0 | 0
x07:  07  |  1        x07:  07       x07:   26   | 0 | 0
x08:  08  |  1        x08:  08       x08:   03   | 0 | 0
x09:  09  |  1        x09:  09       x09:   28   | 0 | 0
x10:  03  |  1        x10:  05       x10:   20   | 0 | 0
x11:  28  |  0        x11:  21       x11:   22   | 0 | 1
x12:  0c  |  1        x12:  0c       x12:   25   | 0 | 0
x13:  0d  |  1        x13:  0d       x13:   0b   | 0 | 0
x14:  0e  |  1        x14:  0e       x14:   27   | 0 | 0
x15:  0f  |  1        x15:  0f       x15:   29   | 0 | 0
x16:  10  |  1        x16:  10       x16:   2f   | 0 | 0
x17:  11  |  1        x17:  11       x17:   2a   | 0 | 0
x18:  12  |  1        x18:  12       x18:   2b   | 0 | 0
x19:  13  |  1        x19:  13       x19:   2c   | 0 | 0
x20:  14  |  1        x20:  14       x20:   2d   | 0 | 0
x21:  15  |  1        x21:  15       x21:   24   | 0 | 0
x22:  16  |  1        x22:  16       x22:   30   | 0 | 0
x23:  17  |  1        x23:  17       x23:   36   | 0 | 0
x24:  18  |  1        x24:  18       x24:   31   | 0 | 0
x25:  19  |  1        x25:  19       x25:   32   | 0 | 0
x26:  1a  |  1        x26:  1a       x26:   33   | 0 | 0
x27:  1b  |  1        x27:  1b       x27:   34   | 0 | 0
x28:  1c  |  1        x28:  1c       x28:   2e   | 0 | 0
x29:  1d  |  1        x29:  1d       x29:   37   | 0 | 0
x30:  1e  |  1        x30:  1e       x30:   3d   | 0 | 0
x31:  1f  |  1        x31:  1f       x31:   38   | 0 | 0
```

```
DCache

x00: 00000000 00000000
x01: 00000000 00000000
x02: 00000000 00000005
x03: 00000000 00000001

x04: 00000000 00000000
x05: 00000000 00000000
x06: 00000000 00000000
x07: 00000000 00000001

x08: 00000000 00000000
x09: 00000000 00000000
x0a: 00000000 00000000
x0b: 00000000 00000002

x0c: 00000000 00000000
x0d: 00000000 00000000
x0e: 00000000 00000000
x0f: 00000000 00000003
```

```
Retire Store Buffer
      value    | v | ptr |   addr
x0: 00000008 | 1 |  1  | 00001028
x1: 00000000 | 0 |     | 00000000
x2: 00000000 | 0 |     | 00000000
x3: 00000000 | 0 |     | 00000000
x4: 00000000 | 0 |     | 00000000
x5: 00000000 | 0 |     | 00000000
x6: 00000000 | 0 |     | 00000000
x7: 00000000 | 0 |     | 00000000

Load Queue Buffer
      value    | v | h | t | s |   addr   | rob_idx
x0: 00000000 | 0 | 1 |   | 1 | 00001020 |    0
x1: 00000000 | 0 |   |   |   | 00001028 |    1
x2: 00000000 | 0 |   | 1 |   | 00000000 |    0
x3: 00000000 | 0 |   |   |   | 00000000 |    0
x4: 00000000 | 0 |   |   |   | 00000000 |    0
x5: 00000000 | 0 |   |   |   | 00000000 |    0
x6: 00000000 | 0 |   |   |   | 00000000 |    0
x7: 00000000 | 0 |   |   |   | 00000000 |    0
```

```
ROB
           rob_idx | T_new | T_old | Inst | valid | br | H | T | ret_tag | h0rdy | h1rdy |    PC    |  ex_tPC  | ex_tbr | pred_tPC | p_tbr | m
rob[0]:      00    |  3c   |  35   | lw   |   1   | 0  | 1 |   |    0    |   0   |   0   | 00000024 | 00000000 |   0    | 00000028 |   0   | 0
rob[1]:      01    |  23   |  3e   | lw   |   1   | 0  |   |   |    0    |       |       | 00000028 | 00000000 |   0    | 0000002c |   0   | 0
rob[2]:      02    |  0a   |  23   | add  |   1   | 0  |   |   |    0    |       |       | 0000002c | 00000000 |   0    | 00000030 |   0   | 0
rob[3]:      03    |  02   |  3f   | addi |   1   | 0  |   |   |    1    |       |       | 00000030 | 00000000 |   0    | 00000034 |   0   | 0
rob[4]:      04    |  26   |  04   | addi |   1   | 0  |   |   |    1    |       |       | 00000034 | 00000000 |   0    | 00000038 |   0   | 0
rob[5]:      05    |  03   |  05   | addi |   1   | 0  |   |   |    0    |       |       | 00000038 | 00000000 |   0    | 0000003c |   0   | 0
rob[6]:      06    |  28   |  21   | slti |   1   | 0  |   |   |    0    |       |       | 0000003c | 00000000 |   0    | 00000040 |   0   | 0
rob[7]:      07    |  00   |  00   | sw   |   1   | 0  |   |   |    0    |       |       | 00000040 | 00000000 |   0    | 00000044 |   0   | 0
rob[8]:      08    |  20   |  06   | addi |   1   | 0  |   |   |    0    |       |       | 00000044 | 00000000 |   0    | 00000048 |   0   | 0
rob[9]:      09    |  00   |  00   | bne  |   1   | 1  |   |   |    0    |       |       | 00000048 | 00000000 |   0    | 00000024 |   0   | 0
rob[10]:     0a    |  00   |  00   | wfi  |   1   | 0  |   |   |    0    |       |       | 0000004c | 00000000 |   0    | 00000050 |   0   | 0
rob[11]:     00    |  00   |  00   | -    |   0   | 0  |   | 1 |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[12]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[13]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[14]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[15]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[16]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[17]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[18]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[19]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[20]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[21]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[22]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[23]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[24]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[25]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[26]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[27]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[28]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[29]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[30]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
rob[31]:     00    |  00   |  00   | -    |   0   | 0  |   |   |    0    |       |       | 00000000 | 00000000 |   0    | 00000000 |   0   | 0
```

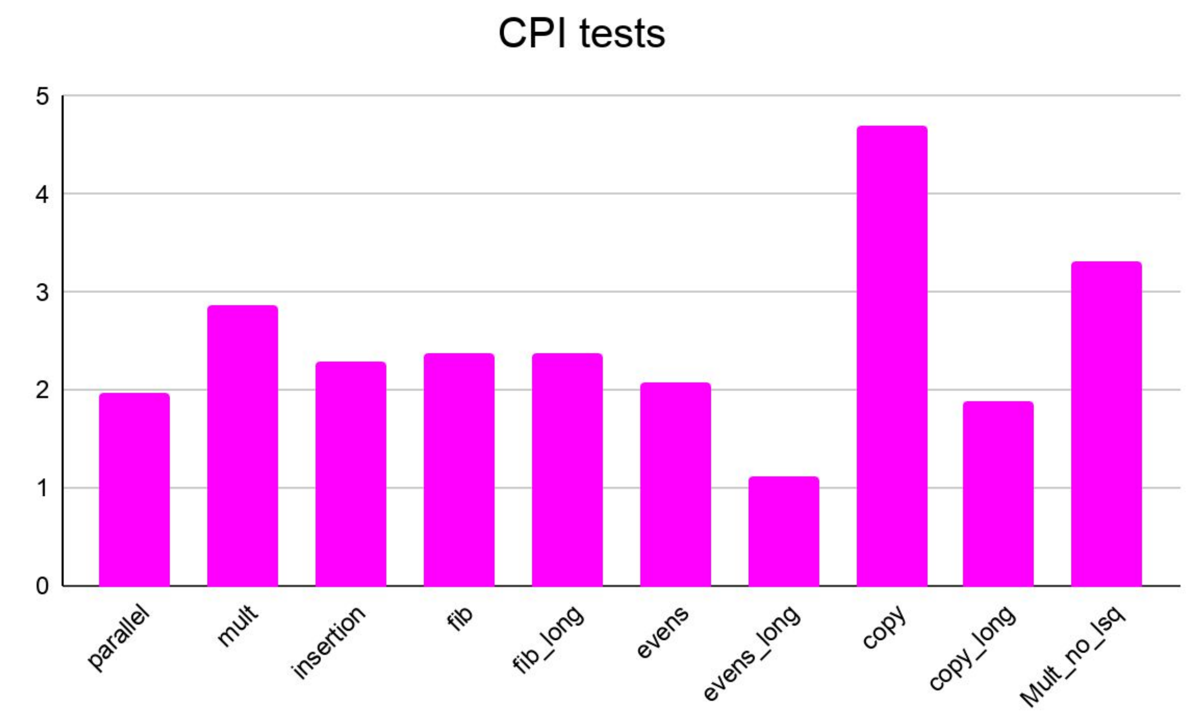
Terminal
sys_defs.svh - group6f20 - Visual St...
[red hat 下怎么截图_百度知道 - Goo...

- **Test on superscalar**

  We tested our superscalar using a simple assembly code without any dependency and the CPI we achieved is 0.61 and the IPC is 1.64. This result is close to what we expected. However when we did tests on provided code, the average CPI was 2.49. The reason for this result might be the number of instructions of these assembly code is small and the penalty of cache miss, miss predict and 8-stage mult is high.

  The assembly code we used to test superscalar is shown below:

```
li x1, 0x0
li x2, 0x0
li x3, 0x0
li x4, 0x0
li x5, 2000
loop:addi x1, x1, 0x1
     addi x2, x2, 0x1
     addi x3, x3, 0x1
     addi x4, x4, 0x1
     bne x1, x5, loop
wfi
```



- **Test on branch predictor**

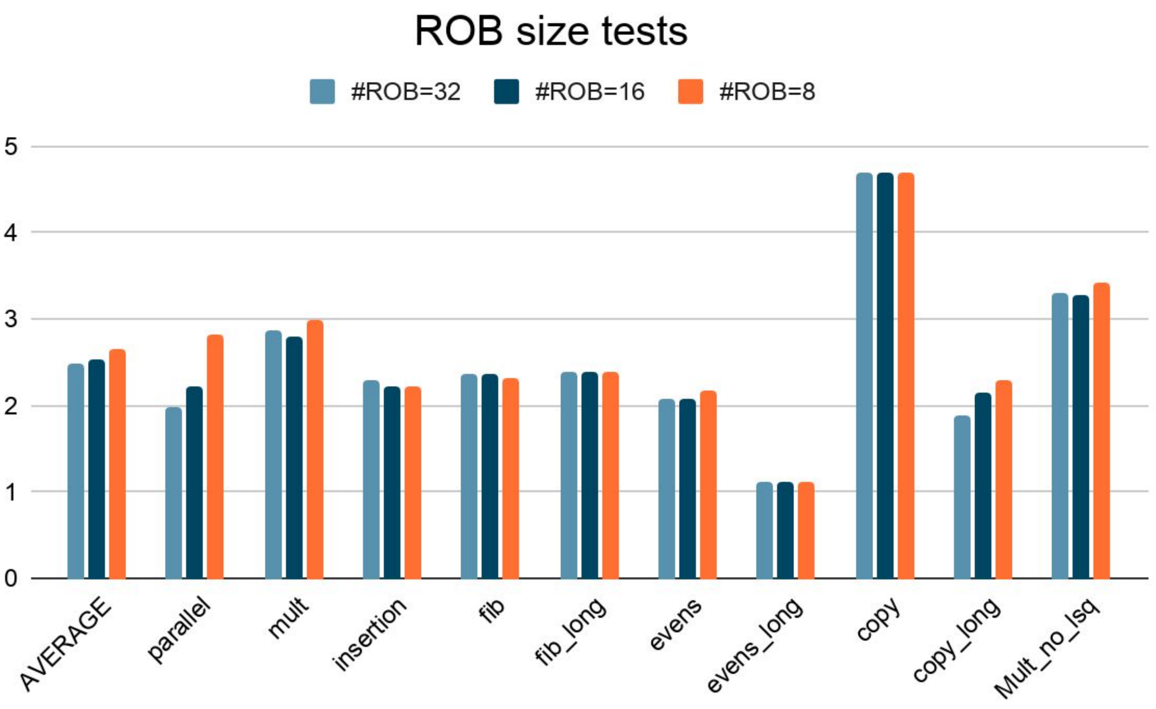
  We tested our branch predictor with a nested loop assembly code. The CPI we achieved is 0.89.

```
li x1, 0xef
li x2, 0xff
loop1: addi x1, x1, -1
loop2: addi x2, x2, -1
      bne x1, x0, loop1
      bne x2, x0, loop2
wfi
```

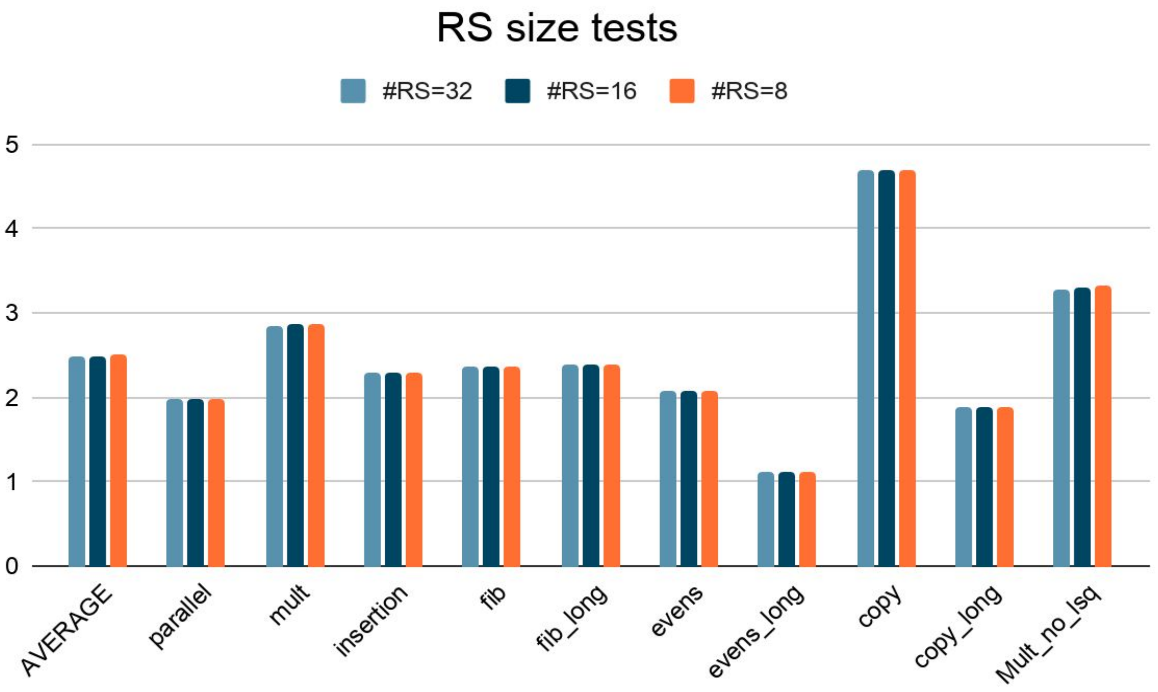
Performance Analysis

- **ROB size**

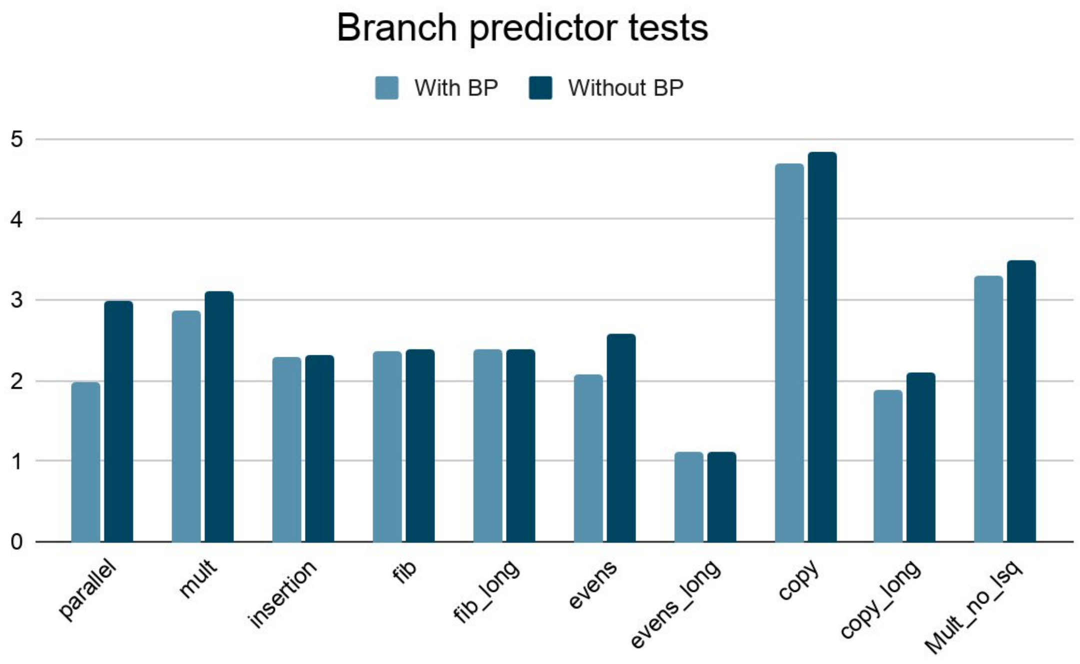
  The number of entries in our ROB is 32. We did tests when ROB size was 16 or 8. For all test programs, the ROB will not be full and will not cause structure hazard if the size is 32. For rv32_parallel, rv32_copy_long, the ROB size has a significant impact on performance. The average CPI is 2.49, 2.53, 2.64 when the number of ROB entries is 32, 16, 8, separately.



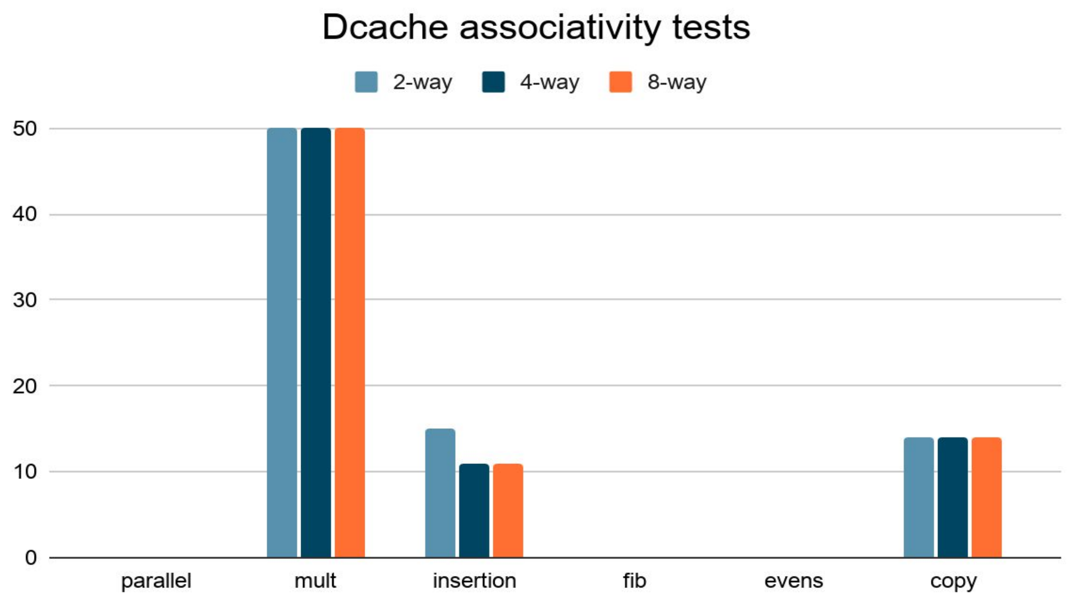
- **RS size**
  We did tests on different RS sizes. We found that for most programs the RS sizes did not affect the CPI. Only for rv32_mult.s and mult_no_lsq.s, the RS will be full and cause structure hazard so the size will slightly influence the performance. The average CPI is 2.49, 2.496, 2.501 when the number of RS entries is 32, 16, 8, separately.



- **Branch prediction**
  We tried testing our system without our branch predictor (predict always not taken). The average CPI without branch predictor is 2.735, which is larger than CPI with branch predictor(2.49). This result shows that our branch predictor effectively improved the system performance.



- **Dcache associativity**
  Our Dcache is parameterized so we tried 2-way, 4-way and 8-way. The chart below shows the eviction times for each program that has load or store instructions. For parallel, fib and evens programs, 16 cache lines is enough so there is no eviction happening. For mult and copy programs different dcache has the same result. The only difference is rv32_insertion.s and 2-way cache will evict 15 times but 4-way and 8-way only evict 11 times.



Summary

In summary, our design achieved 2-way superscalar and other performance improvement as a R10K style out-of-order processor. The advanced features we implemented are listed below. However in our tests, the performance did not achieve what we expected. The reason for that is the penalty of cache miss, branch predict miss and multiple stage of multiplication is high when the code size is relatively small. Also our system still has some bugs which will cause unnecessary Icache miss, so the overall performance is reduced.

- Advanced feature:
  - Superscalar (width = 2)
  - Store-to-load forwarding in LSQ
  - Loads issue out-of-order past pending stores(non-speculative)
  - Post-retirement store buffer
  - Multiple outstanding load misses
  - Next-line prefetching for instructions
  - Write-back data cache
  - Data cache associativity > 1
  - Gshare branch predictor with BTB
  - Return address stack

Contribution

- **Jiajun Cao:** ROB, RS, LSQ, Pipeline Integration, System verification
- **Rui Sun:** ROB, RS, I/D cache, Pipeline Integration, System verification
- **Wenbo Duan:**Maptable, Pipeline Integration, Visual Debugger, System verification
- **Guoqin Yin:** Freelist, LSQ