154B Discussion 6

February 18th, 2022

Goals

- Assignment 4
  - Forwarding and hazard detection for dual issue
  - Graphs
- Cache system stuff.

Logistics

- Extra credits part is added for assignment 4.
  - Goal: improve the performance (in terms of #cycles) of dual-issue

Assignment 4: Forwarding

- 4 sources:
  - pipeA_mem
  - pipeB_mem
  - pipeA_wb
  - pipeB_wb
- 4 destinations:
  - pipeA_ex_rs1
  - pipeA_ex_rs2
  - pipeB_ex_rs1
  - pipeB_ex_rs2

Assignment 4: Forwarding

Commit order:

pipeA_wb
pipeB_wb
pipeA_mem
pipeB_mem

commit earlier

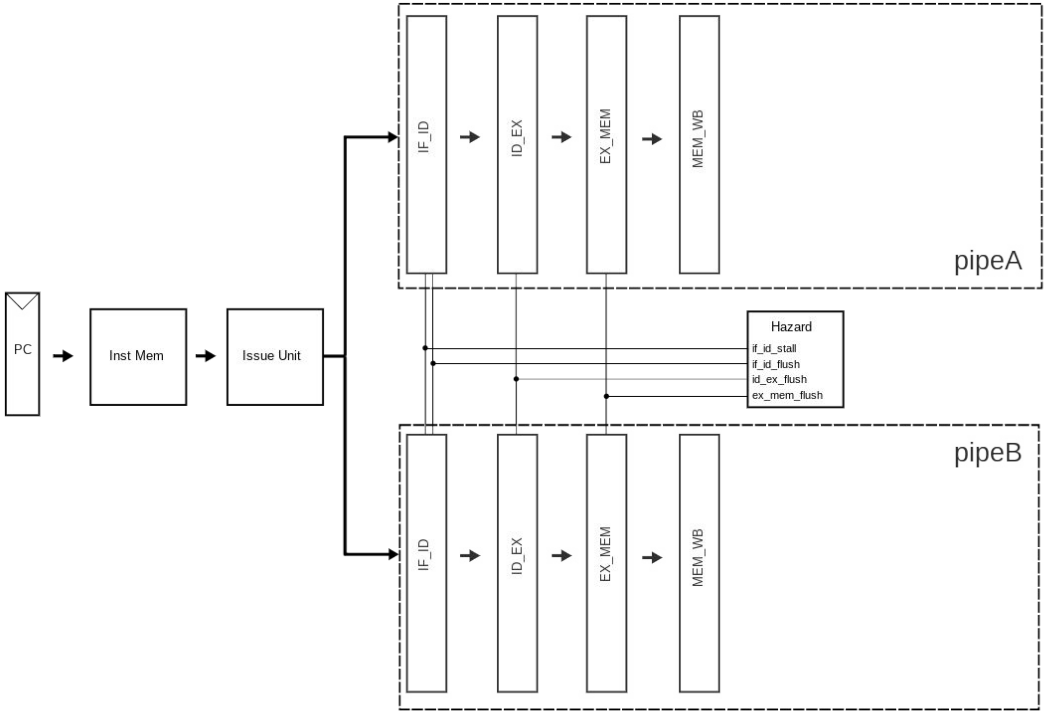
commit later

- 4 sources:
  - pipeA_mem
  - pipeB_mem
  - pipeA_wb
  - pipeB_wb
- Which source should be forwarded if more than one source is available for forwarding?
  - In original pipelined CPU, if we can forward from both MEM and WB, the value from MEM will be used.
  - In dual-issue CPU, at each stage, the instruction in pipeA must be committed before the instruction in pipeB.

Implementing ~~this~~ forwarding to pipeA_ex

when (condition for forwarding from pipeB_mem to pipeA_ex)
{ ... }
elsewhen (condition for forwarding from pipeA_mem to pipeA_ex)
{ ... }
elsewhen ...

Assignment 4: Hazard Detection



Assignment 4: Hazard Detection - Stalling

- Stalling condition: if a load instruction is followed by another instruction reading from the register that the load instruction will write to.

```
ld x1, 1024(x2)

addi x3, x1, x10
```

Assignment 4: Hazard Detection - Stalling

```
ld x1, 1024(x2)

add x3, x1, x10

Original pipeline:

        |  IF |  ID |  EX | MEM |  WB |
        |-----|-----|-----|-----|-----|
Cycle 1 |  LD | --- | --- | --- | --- |
Cycle 2 | ADD |  LD | --- | --- | --- |
Cycle 3 | --- | ADD |  LD | --- | --- |
Cycle 4 | --- | ADD | NOP |  LD | --- |
```

Assignment 4: Hazard Detection - Stalling

- General idea:
  - If the instruction at EX stage is a load instruction, and it writes to the register that the instruction at ID will read, the CPU stalls instruction at ID.
- Stalling detection for dual-issue:
  - There are 2 instructions at EX stage.
  - There are 2 instructions at ID stage.

Assignment 4: Hazard Detection - Stalling

- General idea:
  - If the instruction at EX stage is a load instruction, and it writes to the register that the instruction at ID will read, the CPU stalls instruction at ID.
- Stalling detection for dual-issue:
  - There are 2 instructions at EX stage.
  - There are 2 instructions at ID stage.
  - 4 cases:
    - pipeA_ex is a load instruction, it writes to a register that pipeA_id will read.
    - pipeA_ex is a load instruction, it writes to a register that pipeB_id will read.
    - pipeB_ex is a load instruction, it writes to a register that pipeA_id will read.
    - pipeB_ex is a load instruction, it writes to a register that pipeB_id will read.
  - Why? Instructions at ID stage must be committed after instructions at EX stage.

Assignment 4: Stalling due to load in pipeA

```
              |  IF |  ID |  EX | MEM |  WB |
              |-----|-----|-----|-----|-----|
Cycle 1 pipeA |  LD | --- | --- | --- | --- |
Cycle 1 pipeB | --- | --- | --- | --- | --- |
              |-----------------------------|
Cycle 2 pipeA | ADD |  LD | --- | --- | --- |
Cycle 2 pipeB | SUB | --- | --- | --- | --- |
              |-----------------------------|
Cycle 3 pipeA | --- | ADD |  LD | --- | --- |
Cycle 3 pipeB | --- | SUB | --- | --- | --- |
              |-----------------------------|
Cycle 4 pipeA | --- | ADD | NOP |  LD | --- |
Cycle 4 pipeB | --- | SUB | NOP | --- | --- |
```

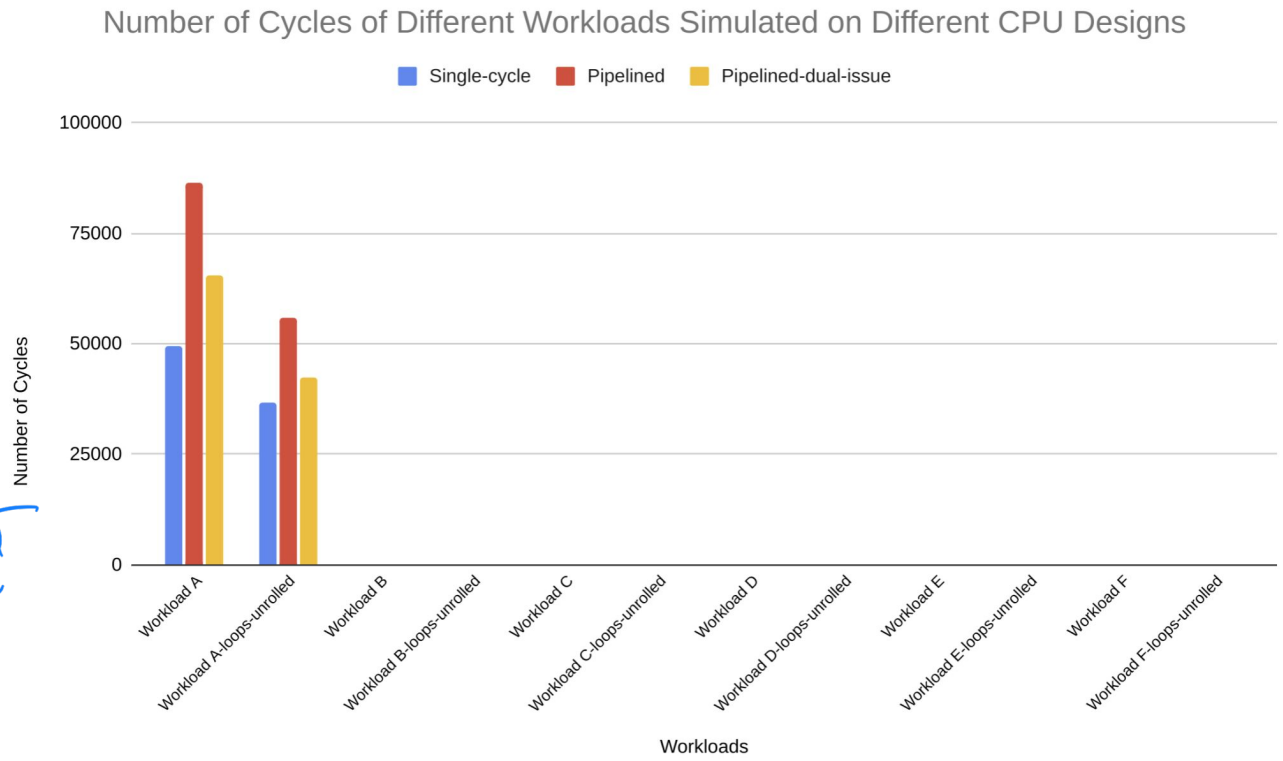
Assignment 4: Stalling due to load in pipeB

```
              |  IF |  ID |  EX | MEM |  WB |
              |-----|-----|-----|-----|-----|
Cycle 1 pipeA | SRA | --- | --- | --- | --- |
Cycle 1 pipeB |  LD | --- | --- | --- | --- |
              |-----------------------------|
Cycle 2 pipeA | ADD | SRA | --- | --- | --- |
Cycle 2 pipeB | SUB |  LD | --- | --- | --- |
              |-----------------------------|
Cycle 3 pipeA | --- | ADD | SRA | --- | --- |
Cycle 3 pipeB | --- | SUB |  LD | --- | --- |
              |-----------------------------|
Cycle 4 pipeA | --- | ADD | NOP | SRA | --- |
Cycle 4 pipeB | --- | SUB | NOP |  LD | --- |
```

Assignment 4: Hazard Detection - Flushing

- If the branch at either instructions at MEM stage was mispredicted, the instructions at ID, EX must be flushed.

Assignment 4: Example Graph



y-axis should have the unit of data

Assignment 4: Graphs

- Graphs should be easy to interpret
  - Labels/Units on both x-axis and y-axis
  - Question 1: the y-axis should be IPC
  - Question 2: the y-axis should be time (in seconds)

Cache System

- general ideas:
  - Every cache block must have all information to identify to which address the data in a cache block belongs to.
    - → every cache block has a tag (contain part of address) associated with it
    - → the index of the cache block indicates on which row the cache block is
    - → the offset indicates where the data is in a cache block.

Address

TAG	INDEX	OFFSET

A cache block w/ tag

TAG	Cache_block_K

Cache System

Direct-mapped cache

TAG	cache_block_0	row 0
TAG	cache_block_1	row 1
...	...	⋮
...	...	
...	...	
TAG	Cache_block_K-1	row K-1

1. The index bits are used to select a row
   → if there are K cache_blocks, there are K rows
   → # index bits = $\log_2(K)$

2. Offset bits: where the data is within a block
   → # offset bits = $\log_2$(block_size in bytes)

Byte 0	Byte 1	...	Byte N-1

assume byte-addressable

3. Tag bits: the rest of the address (needed to identify the mem address of the cache block)

tag bits = address size − # offset bits − # index bits

Cache System

N-ways set associative
(each row has N cache block)

TAG	$_block	TAG	$_block	. . .	TAG	$_block	row 0
TAG	$_block	TAG	$_block	. . .	TAG	$_block	row 1
							row $\frac{K}{N}$

* Assume there are K cache blocks.

→ there are $\frac{K}{N}$ rows

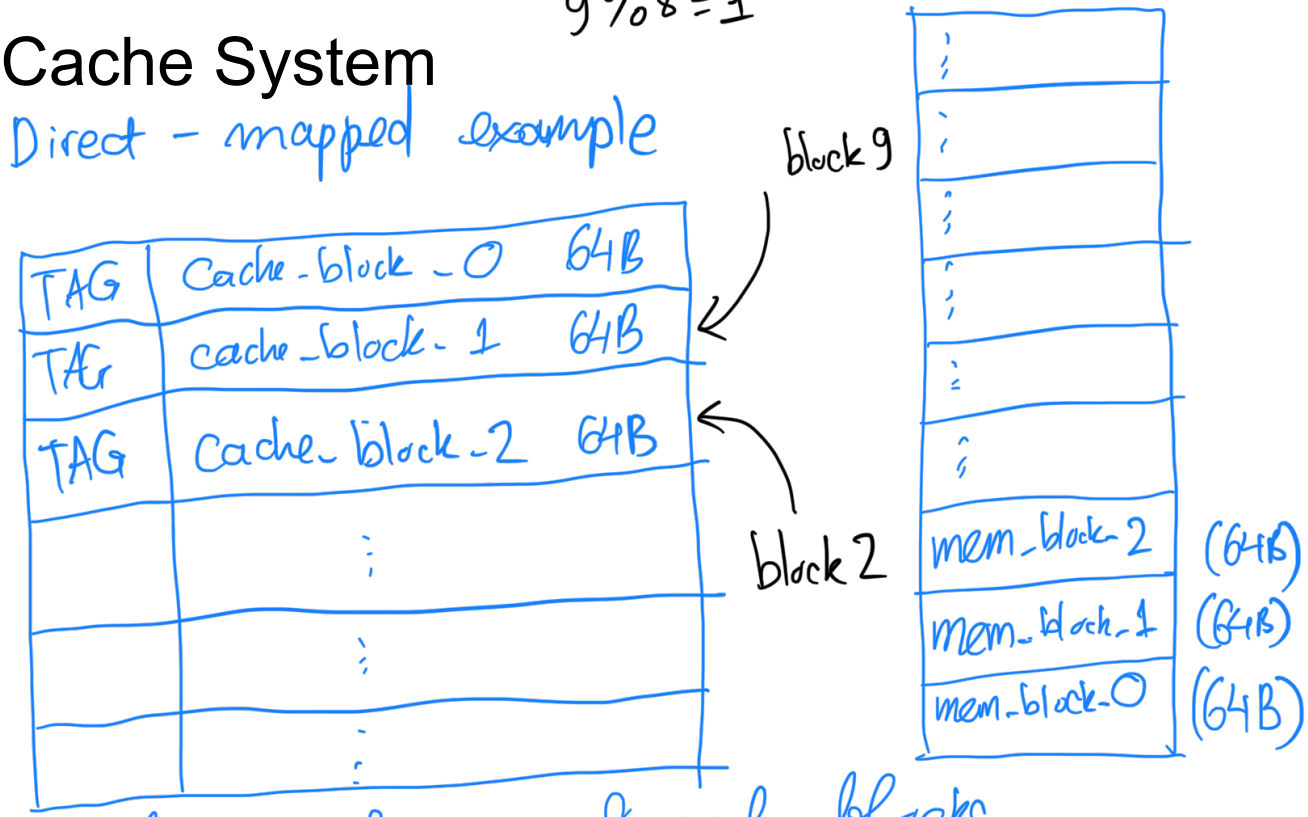
→ # index bits $= \log_2\left(\frac{K}{N}\right)$

- # Offset bits
  $= \log_2(\text{cache\_block\_size})$
- # Tag bits
  = address size − # offset bits
  − # index bits

Cache System

Direct - mapped example

$9 \% 8 = 1$



* Assume there are 8 cache blocks.

→ memory block K will go to cache_block_(K % 8)

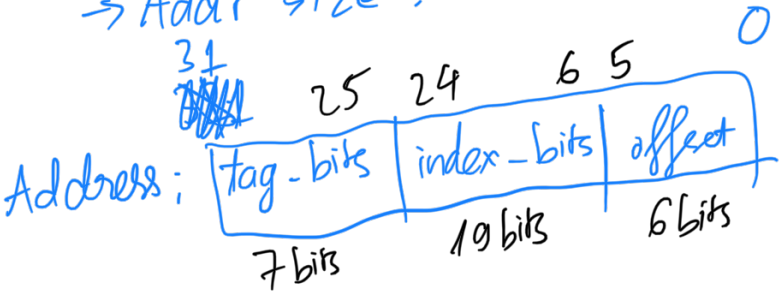
Cache System

Direct-mapped cache example

→ Data capacity : 32768 KiB → 2^{25} bytes

→ block size : 64B (2^6 bytes)

→ Addr size : 32 bits



offset : (5, 0)

index : (24, 6)

tag : (31, 25)

$$\#\ \text{cache\_blocks} = \frac{\text{data\_capacity}}{\text{block\_size}} = \frac{2^{25}}{2^6} = 2^{19}$$

$$\#\ \text{index\_bits} = \log_2(\#\ \text{rows}) = \log_2(\#\ \text{cache\_block}) = \log_2(2^{19}) = 19$$

$$\#\ \text{offset\_bits} = \log_2(\text{block\_size}) = \log_2(2^6) = 6$$

$$\#\ \text{tag\_bits} = \text{addr\_size} - \#\ \text{index\_bits} - \#\ \text{offset\_bits} = 32 - 19 - 6 = 7$$

Cache System

Q4: SystemA : cache block of 16B → 4 bits for offset

TAG	DATA

0x3214 M [3210 - 321f]

0x31e8 M [31e0 - 31ef]

0x31e8 H

0x3220 M [3220 - 322f]

0x31dc M [31d0 - 31df]

SystemB : cache block of 256B → 8 bits for offset

TAG	DATA

0x3214 M [3200 - 32ff]

0x31e8 M [3100 - 31ff]

0x31e8 H

0x3220 H

Cache System