

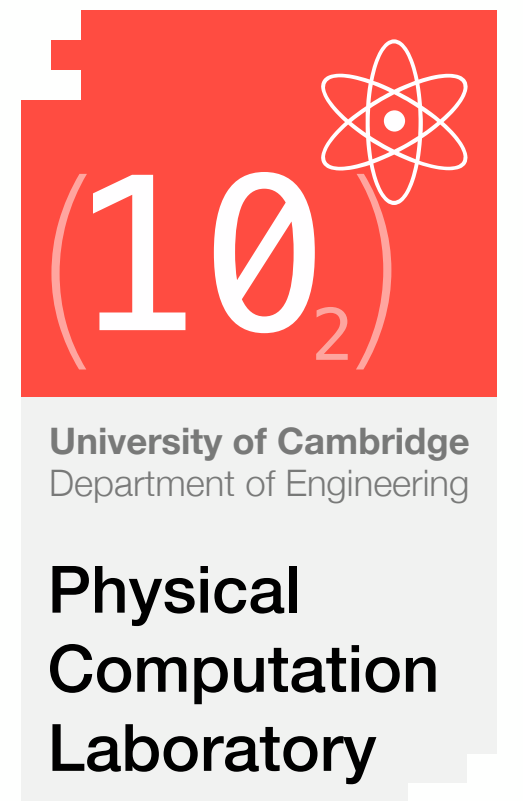
# RISC-V Processor Design

## Fascicle 5: Pipelining

(15 minutes)  
May 2019

**Phillip Stanley-Marbell**

Department of Engineering, University of Cambridge  
<http://physcomp.eng.cam.ac.uk>



# Intended Learning Outcomes for Today

---

2

12

**By the end of this session, you should be able to:**

# Intended Learning Outcomes for Today

---

**By the end of this session, you should be able to:**

- ➊ Identify a pipelined microarchitecture and its pipeline stages

# Intended Learning Outcomes for Today

---

**By the end of this session, you should be able to:**

- ❶ Identify a pipelined microarchitecture and its pipeline stages
- ❷ Propose how to split a given microarchitecture into pipeline stages

# Intended Learning Outcomes for Today

---

**By the end of this session, you should be able to:**

- ❶ Identify a pipelined microarchitecture and its pipeline stages
- ❷ Propose how to split a given microarchitecture into pipeline stages
- ❸ Estimate achievable clock speed and instruction throughput for a pipelined design

# Intended Learning Outcomes for Today

---

**By the end of this session, you should be able to:**

- ❶ Identify a pipelined microarchitecture and its pipeline stages
- ❷ Propose how to split a given microarchitecture into pipeline stages
- ❸ Estimate achievable clock speed and instruction throughput for a pipelined design
- ❹ Enumerate properties of an instruction set that make it well-suited to pipelining

# Pipelining Improves Throughput

---

# Pipelining Improves Throughput

---

**Pipelining is a form of parallelism**



# Pipelining Improves Throughput

---

## **Pipelining is a form of parallelism**

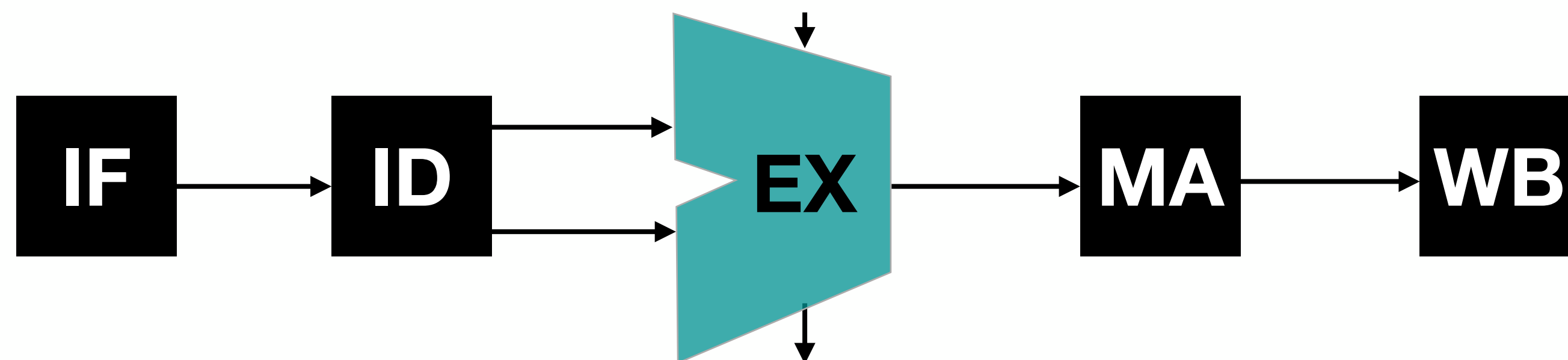
At any point in time, there are different instructions in different phases of execution

# Pipelining Improves Throughput

## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)



# Pipelining Improves Throughput

3

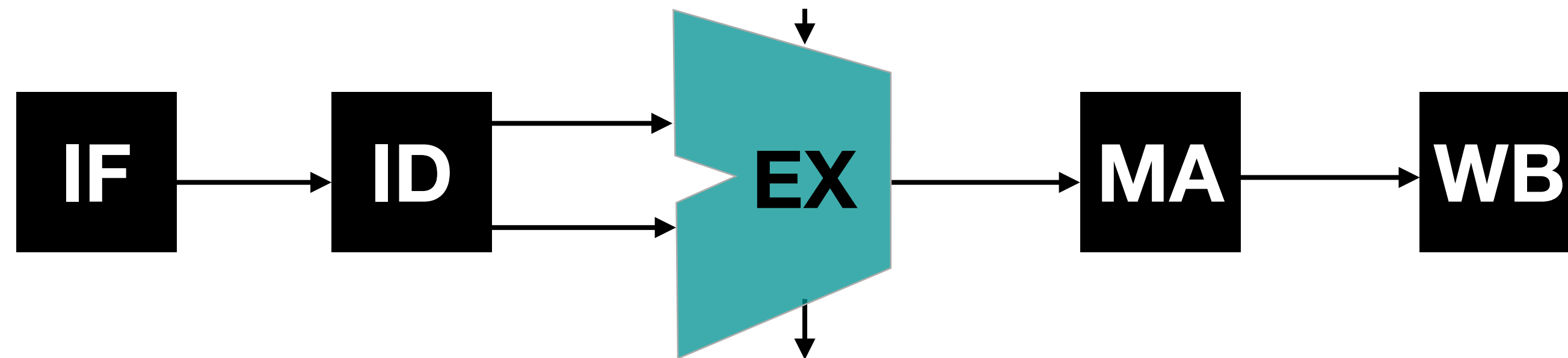
12

## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example



# Pipelining Improves Throughput

3

12

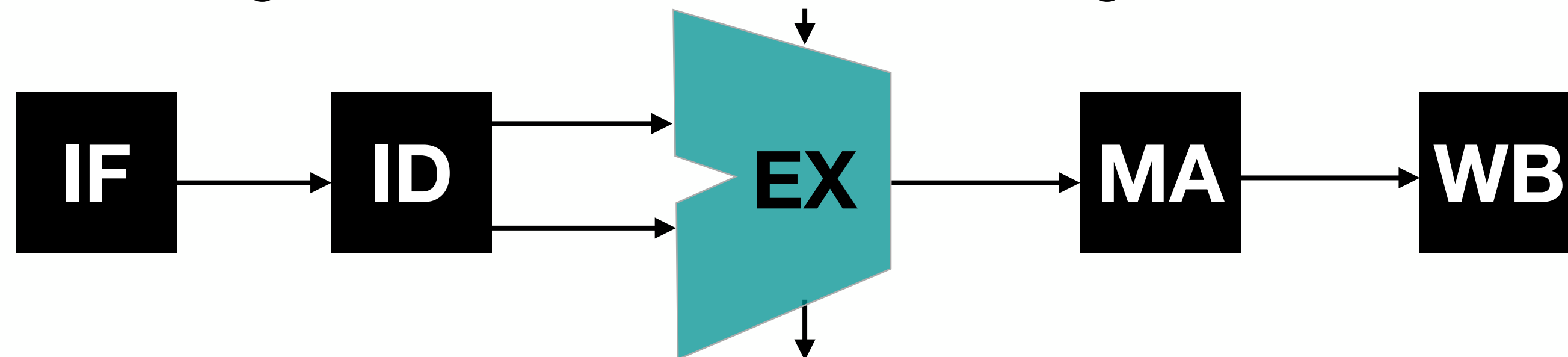
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



# Pipelining Improves Throughput

3

12

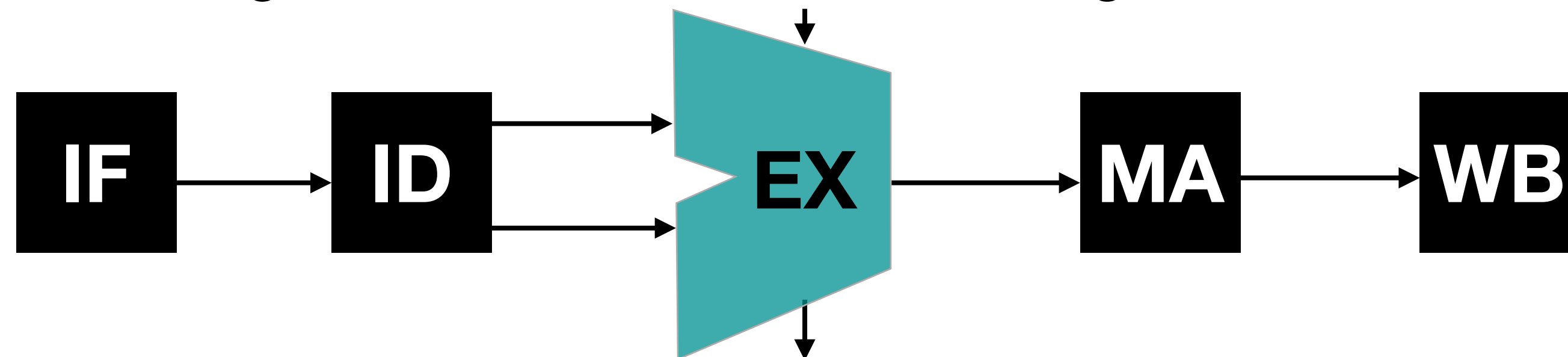
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

# Pipelining Improves Throughput

3

12

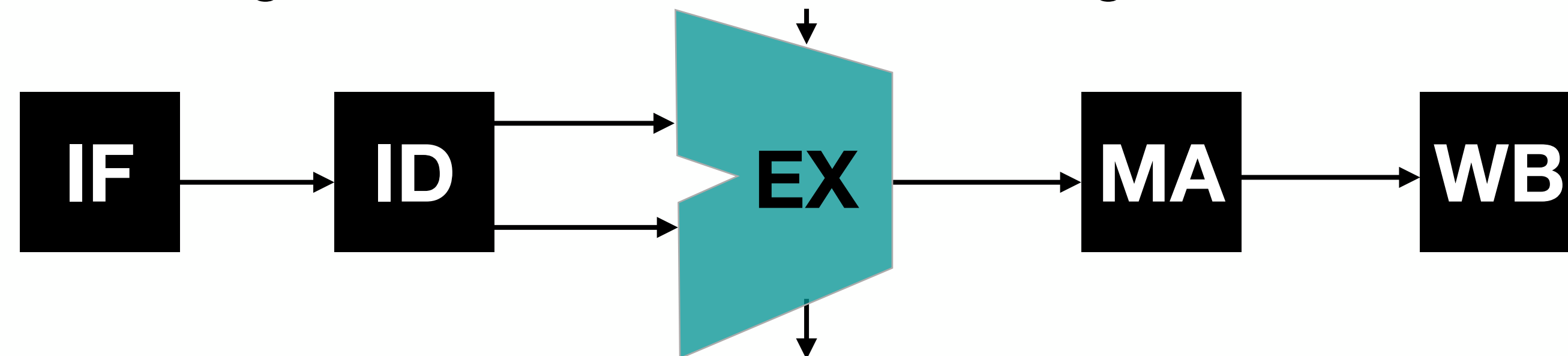
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time

# Pipelining Improves Throughput

3

12

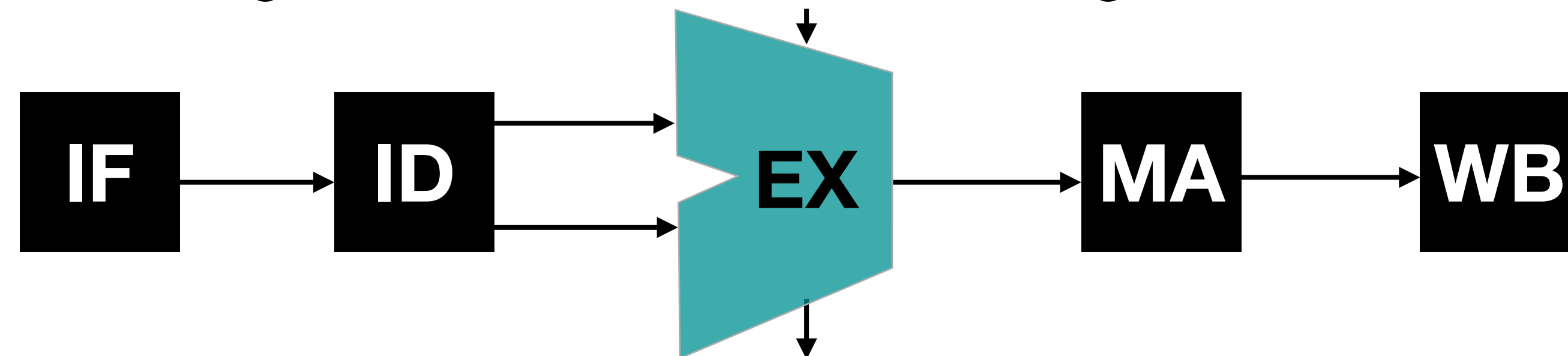
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time

# Pipelining Improves Throughput

3

12

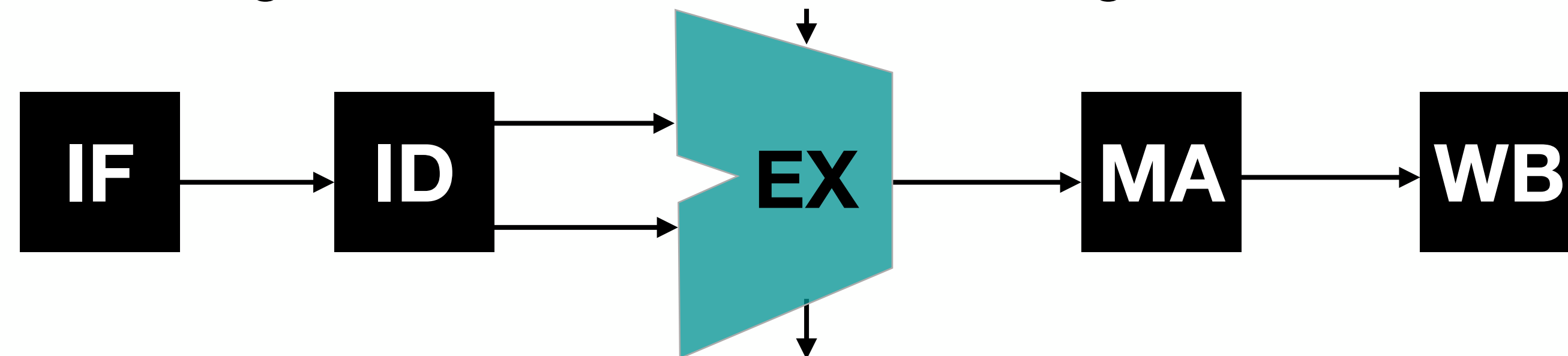
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

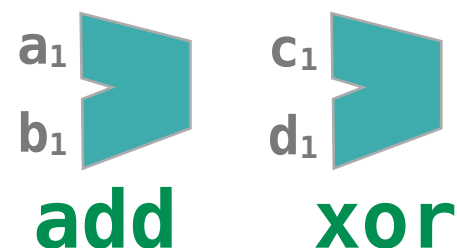
## Example

While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time





# Pipelining Improves Throughput

3

12

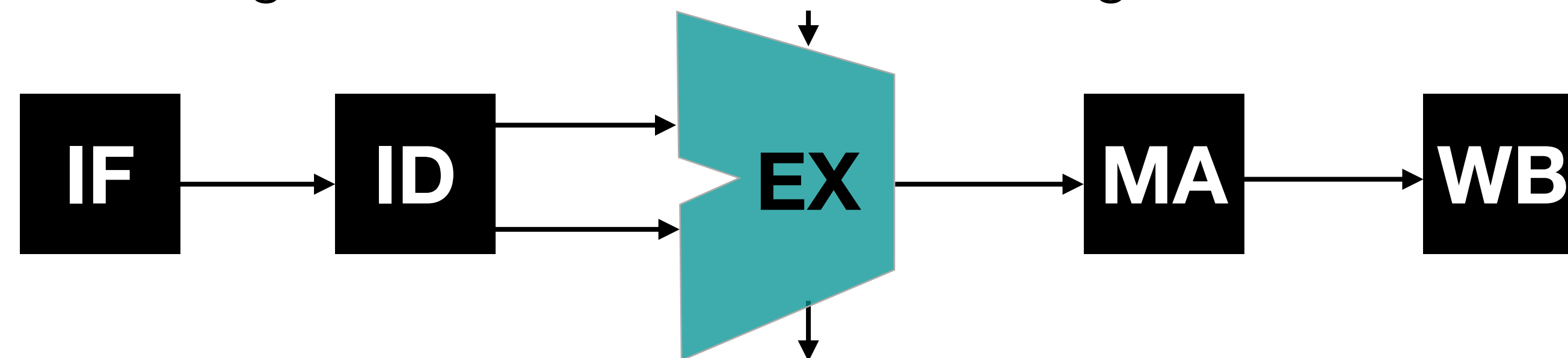
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

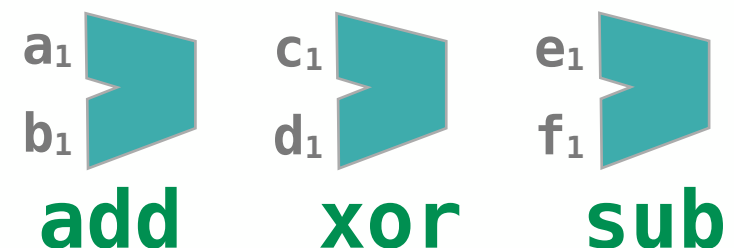
## Example

While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time



# Pipelining Improves Throughput

3

12

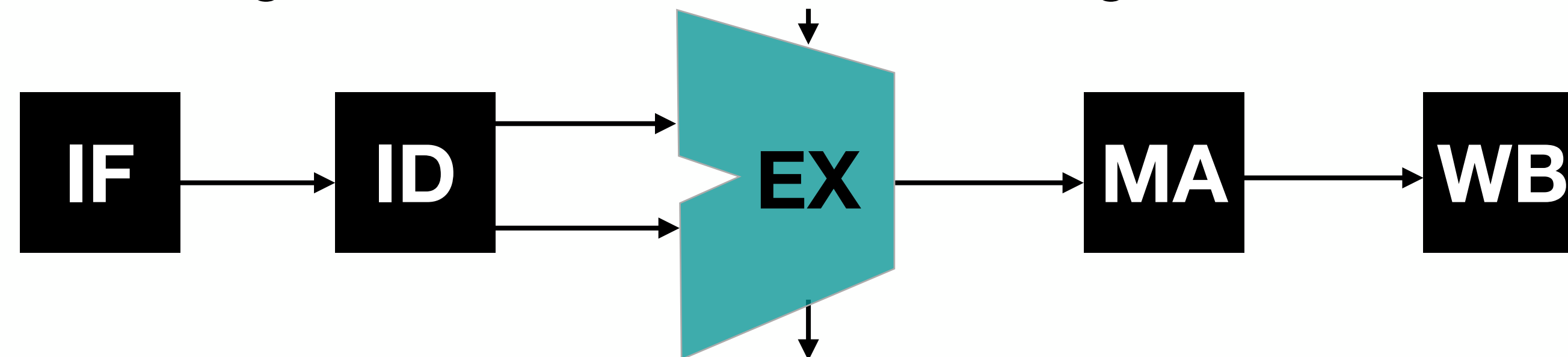
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

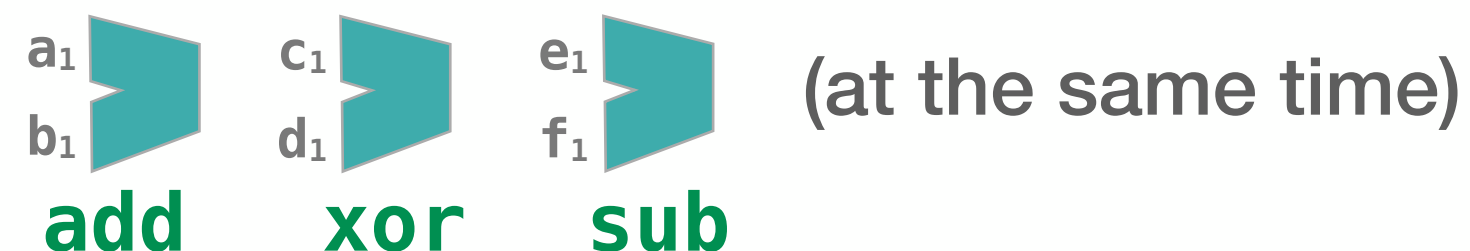
## Example

While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time



# Pipelining Improves Throughput

3

12

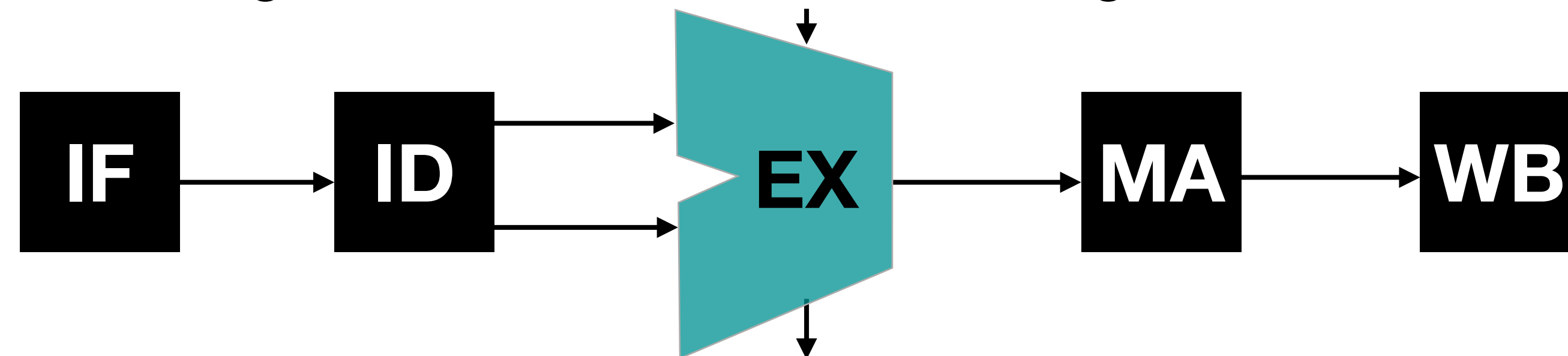
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

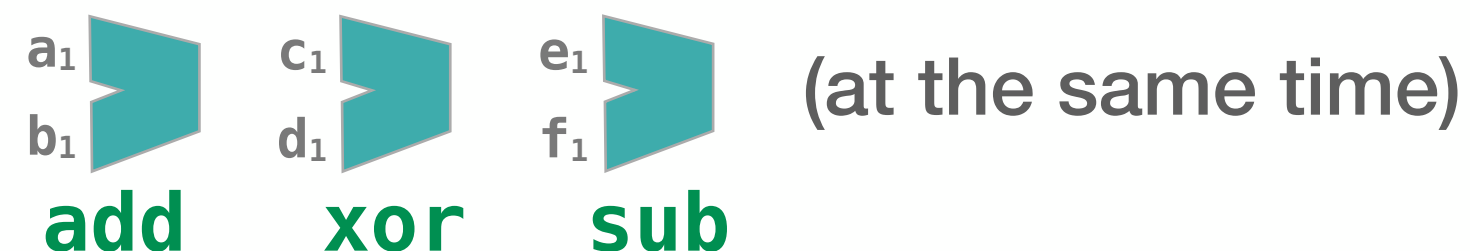
While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time

**Vector:** multiple data in operation



# Pipelining Improves Throughput

3

12

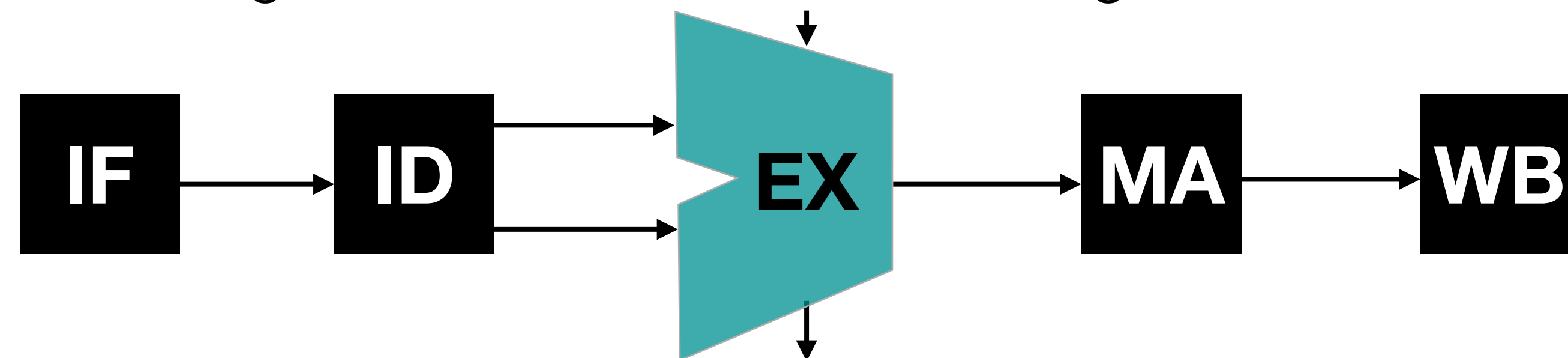
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

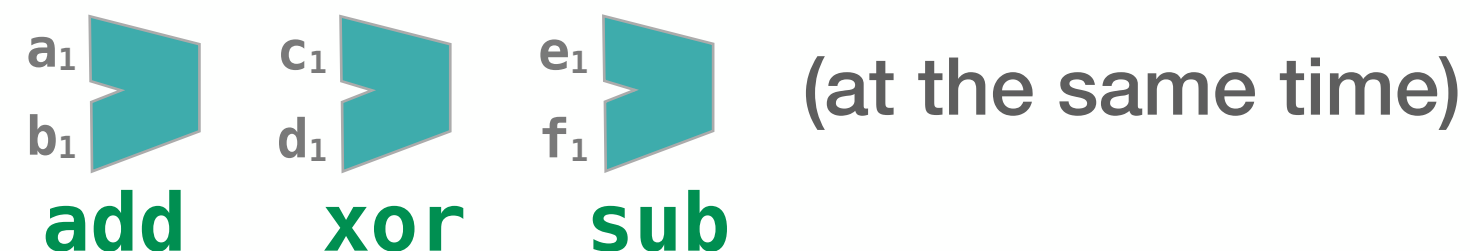
While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time

**Vector:** multiple data in operation



**add**

# Pipelining Improves Throughput

3

12

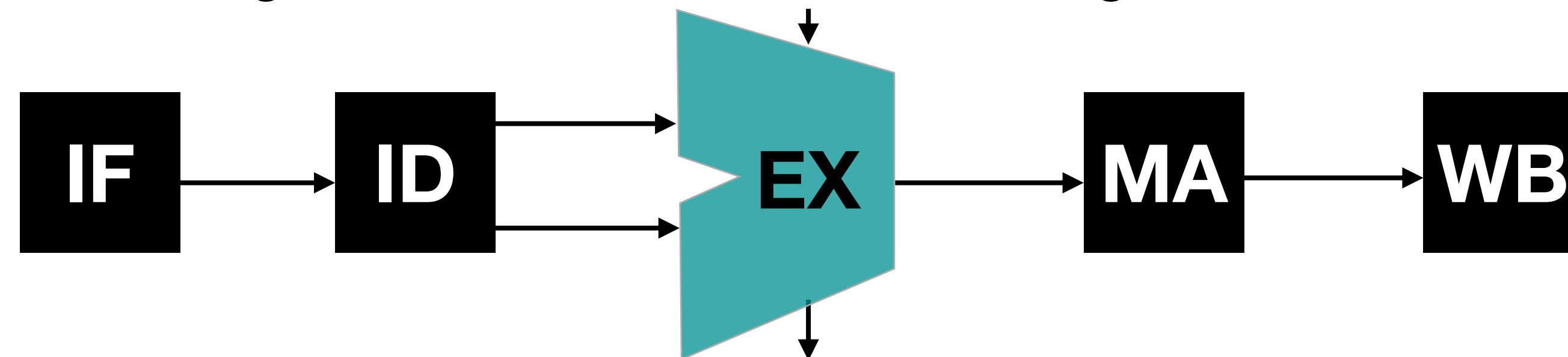
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

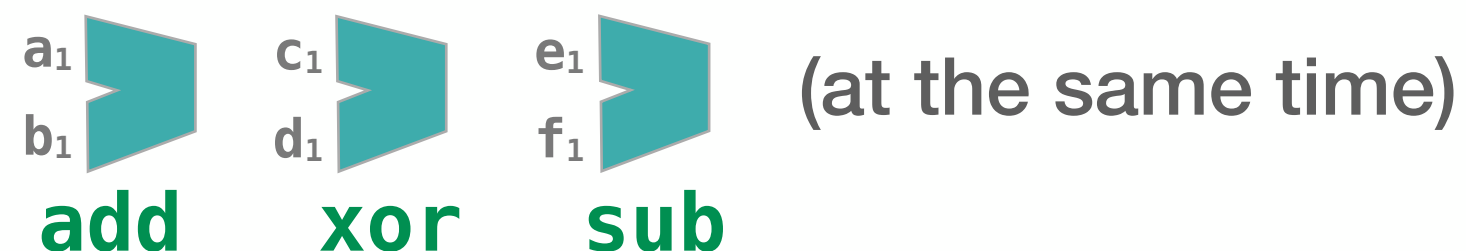
While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time

**Vector:** multiple data in operation



# Pipelining Improves Throughput

3

12

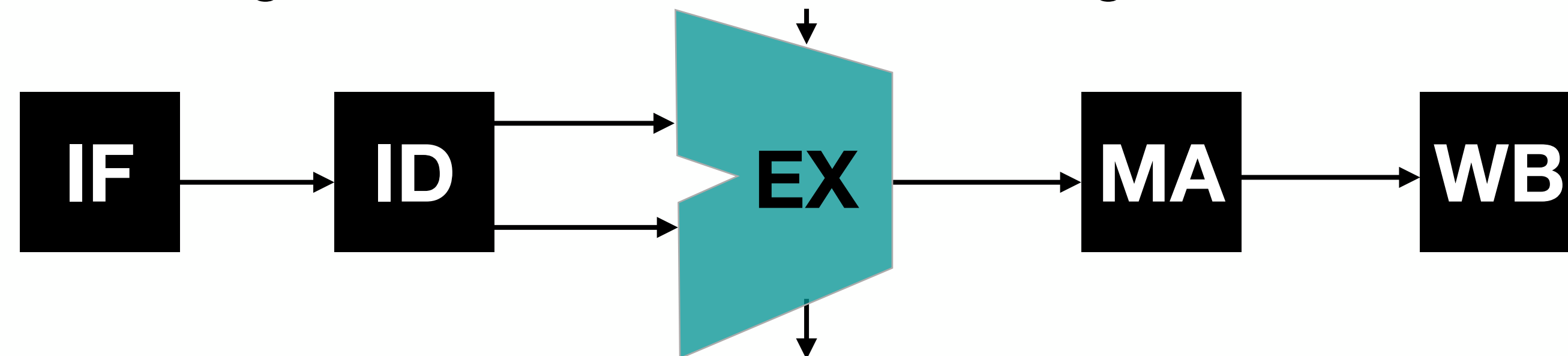
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

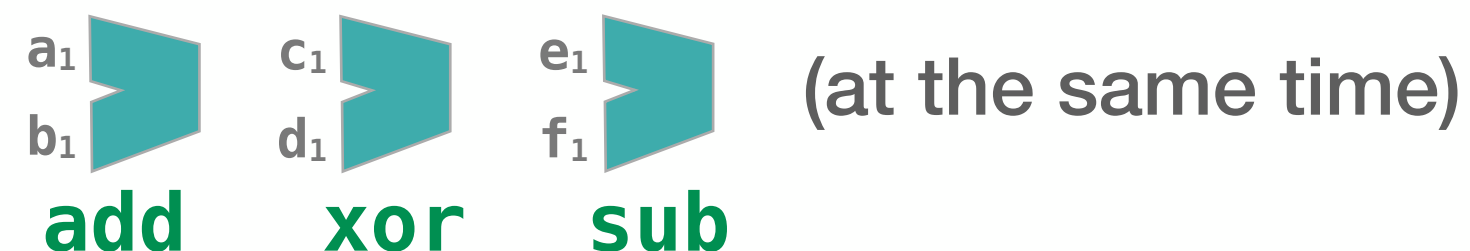
While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time

**Vector:** multiple data in operation



# Pipelining Improves Throughput

3

12

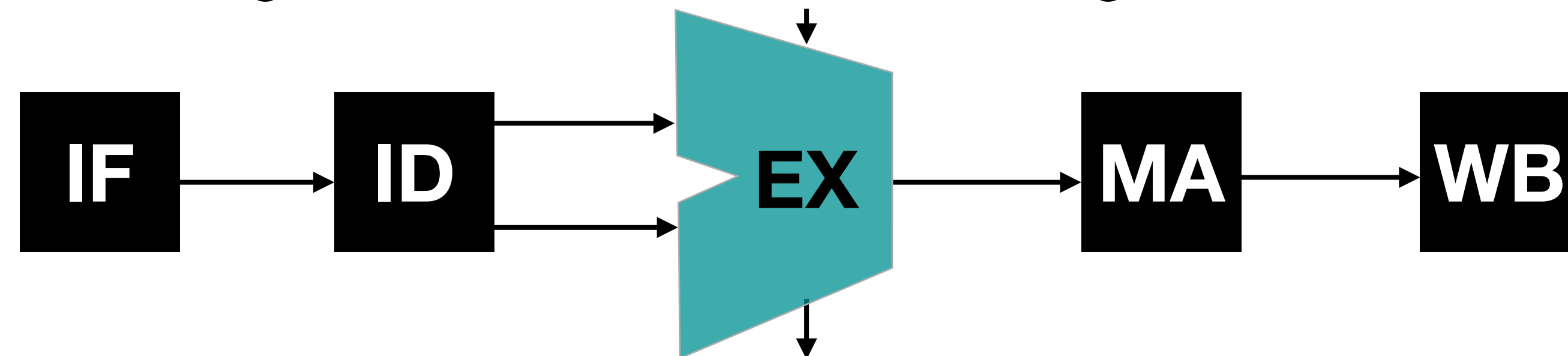
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

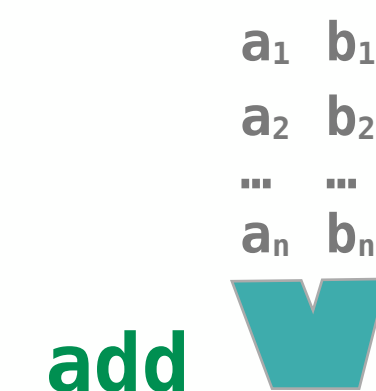
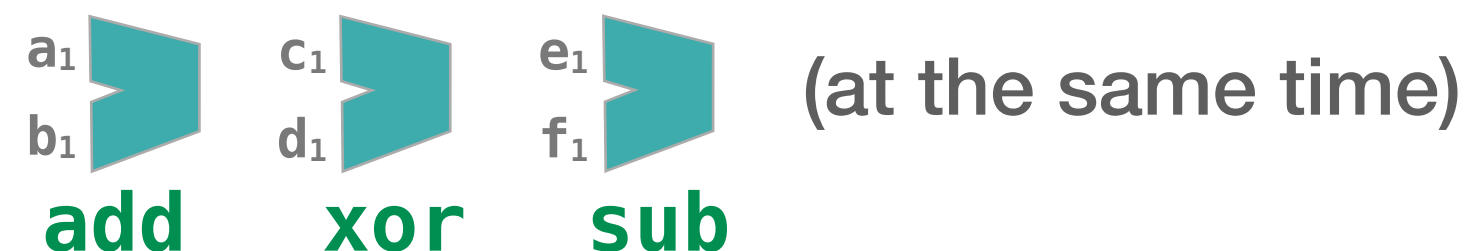
While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time

**Vector:** multiple data in operation





# Pipelining Improves Throughput

3

12

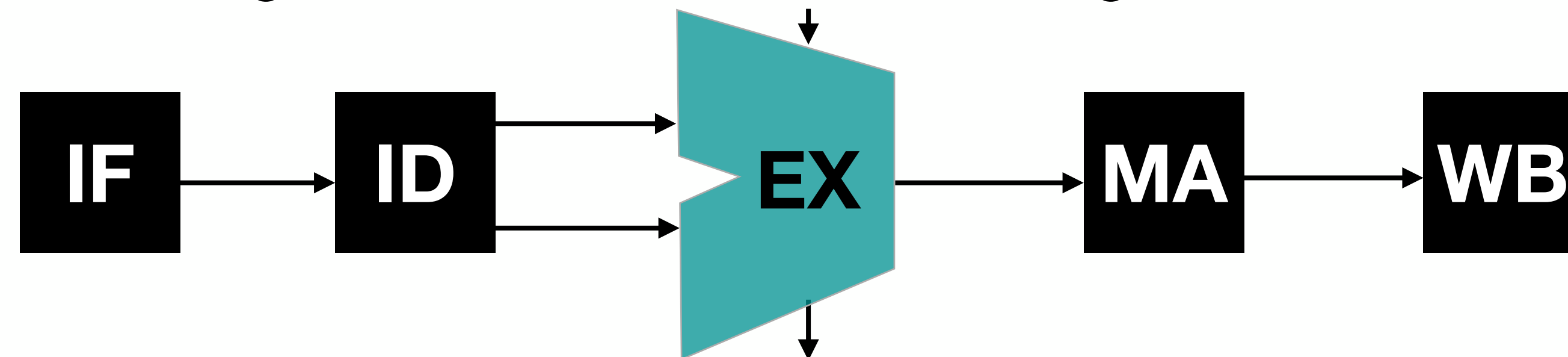
## Pipelining is a form of parallelism

At any point in time, there are different instructions in different phases of execution

Recall, **Lecture 1** (Slide 10) and **Fascicle 1** (Slide 11)

## Example

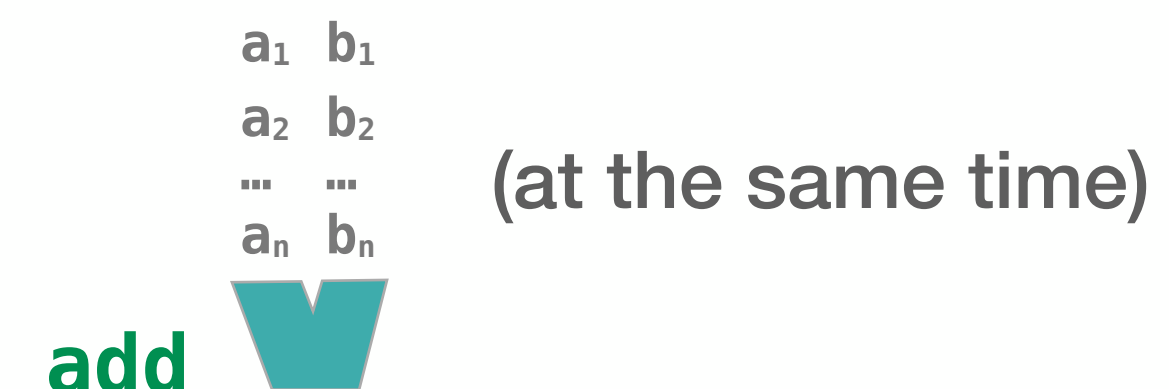
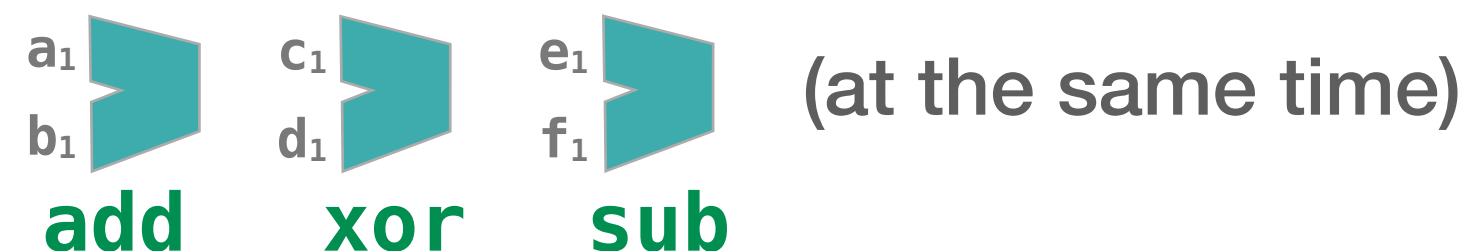
While an instruction is executing, the instruction after it is being decoded, the next is being fetched, etc.



## Contrast pipelining with superscalar and vector execution

**Superscalar:** multiple (different) instructions at same time

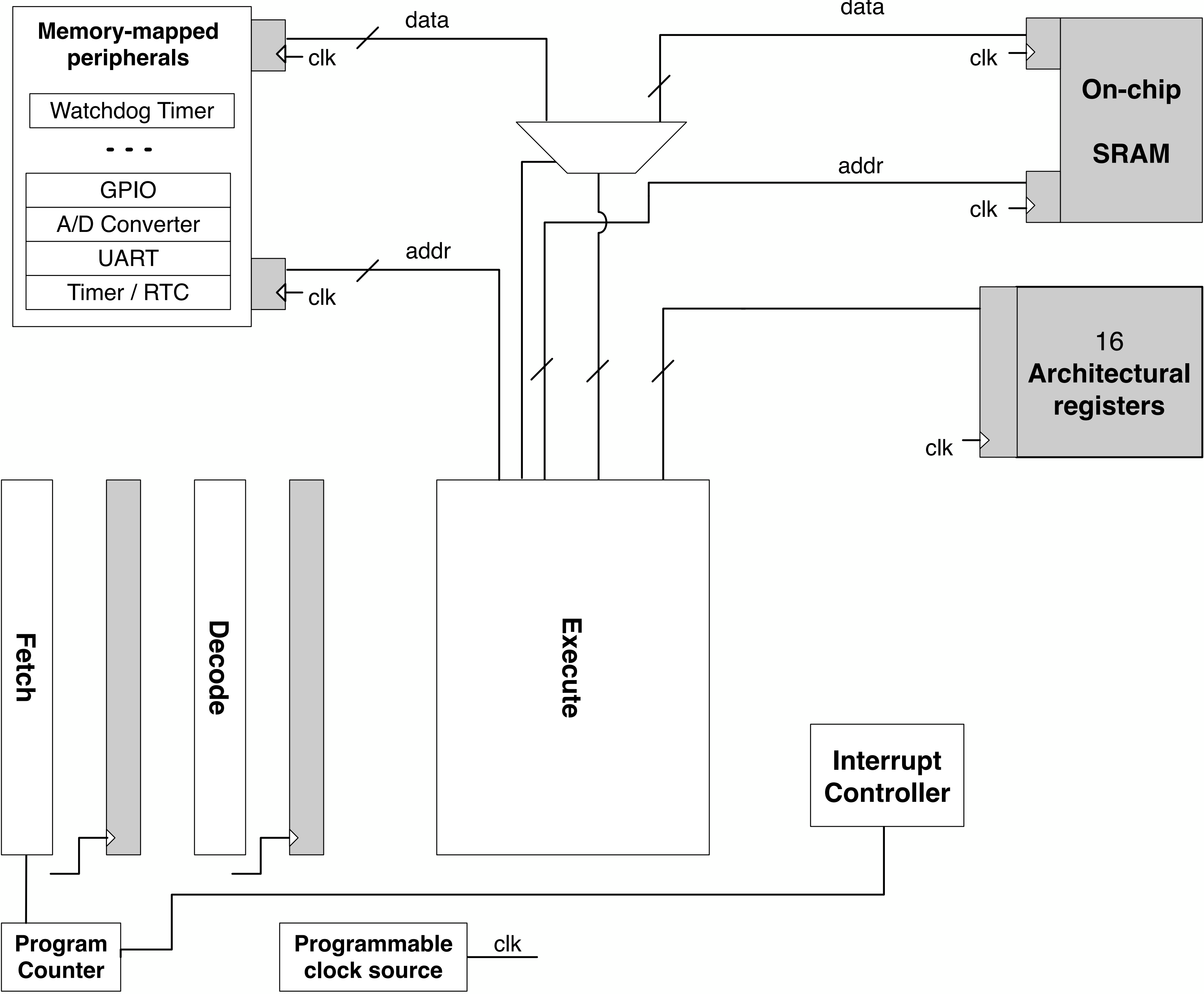
**Vector:** multiple data in operation





4  
12

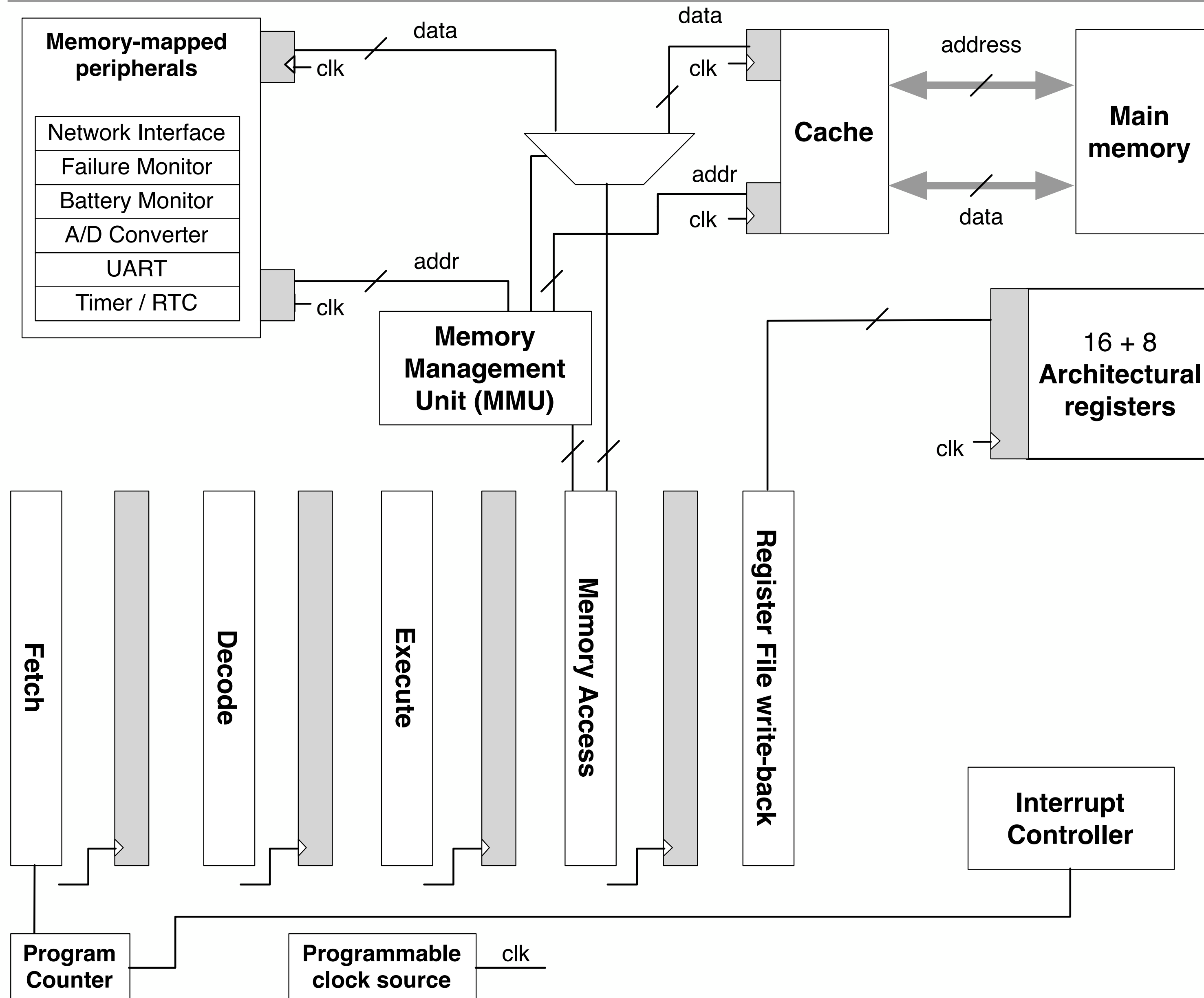


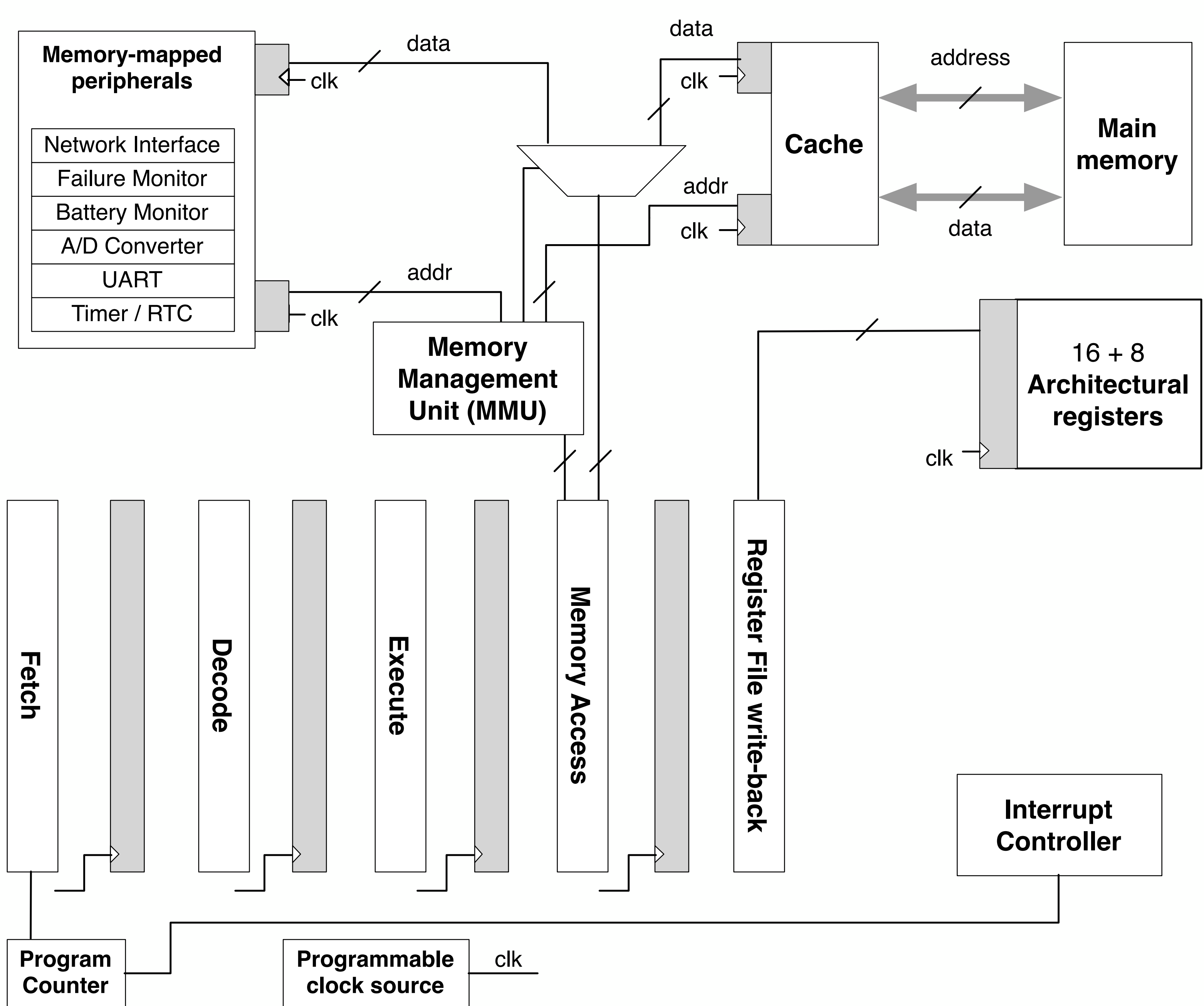


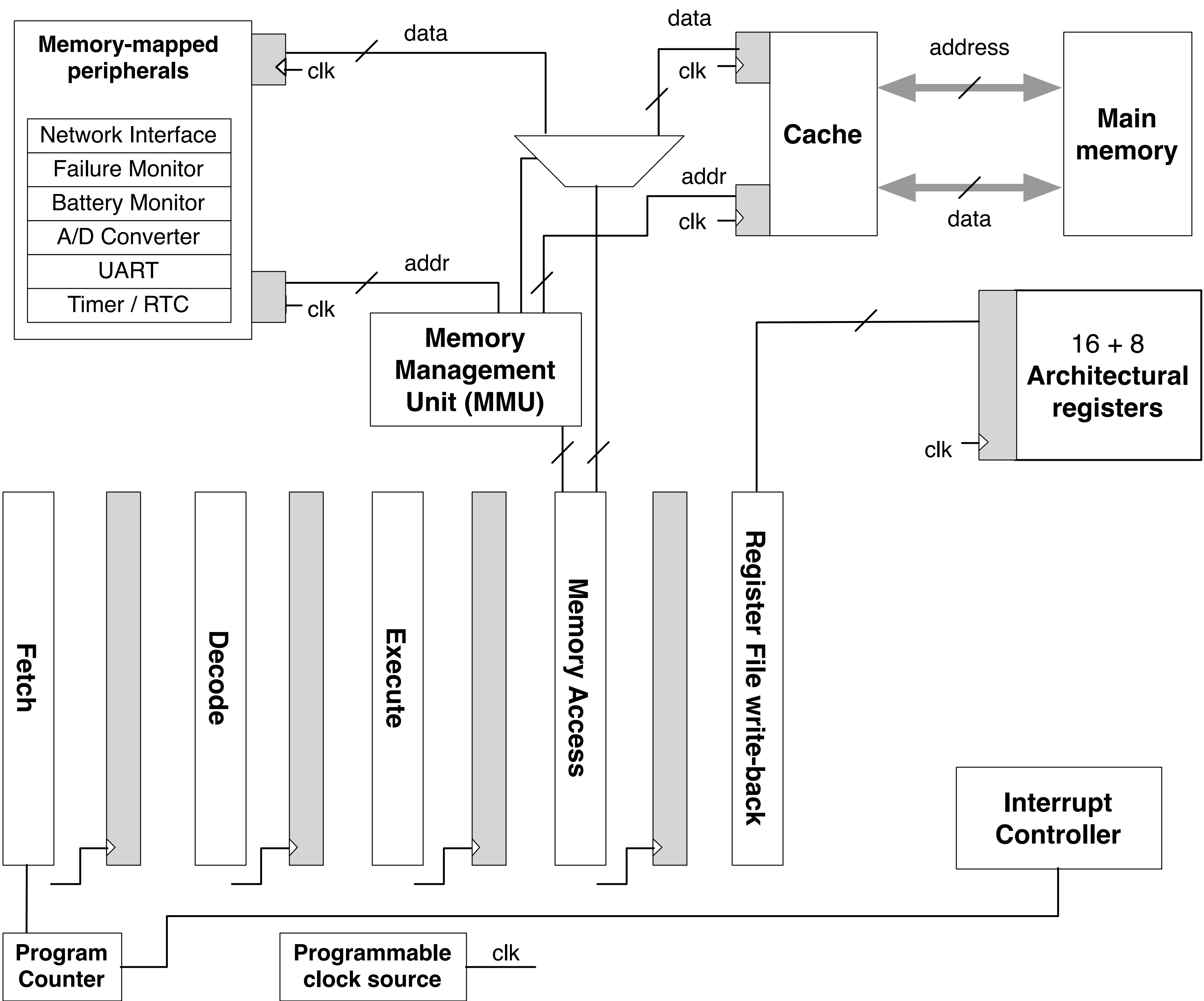
# Example: Hitachi SH3 SH7708 5-Stage Pipeline

6

12







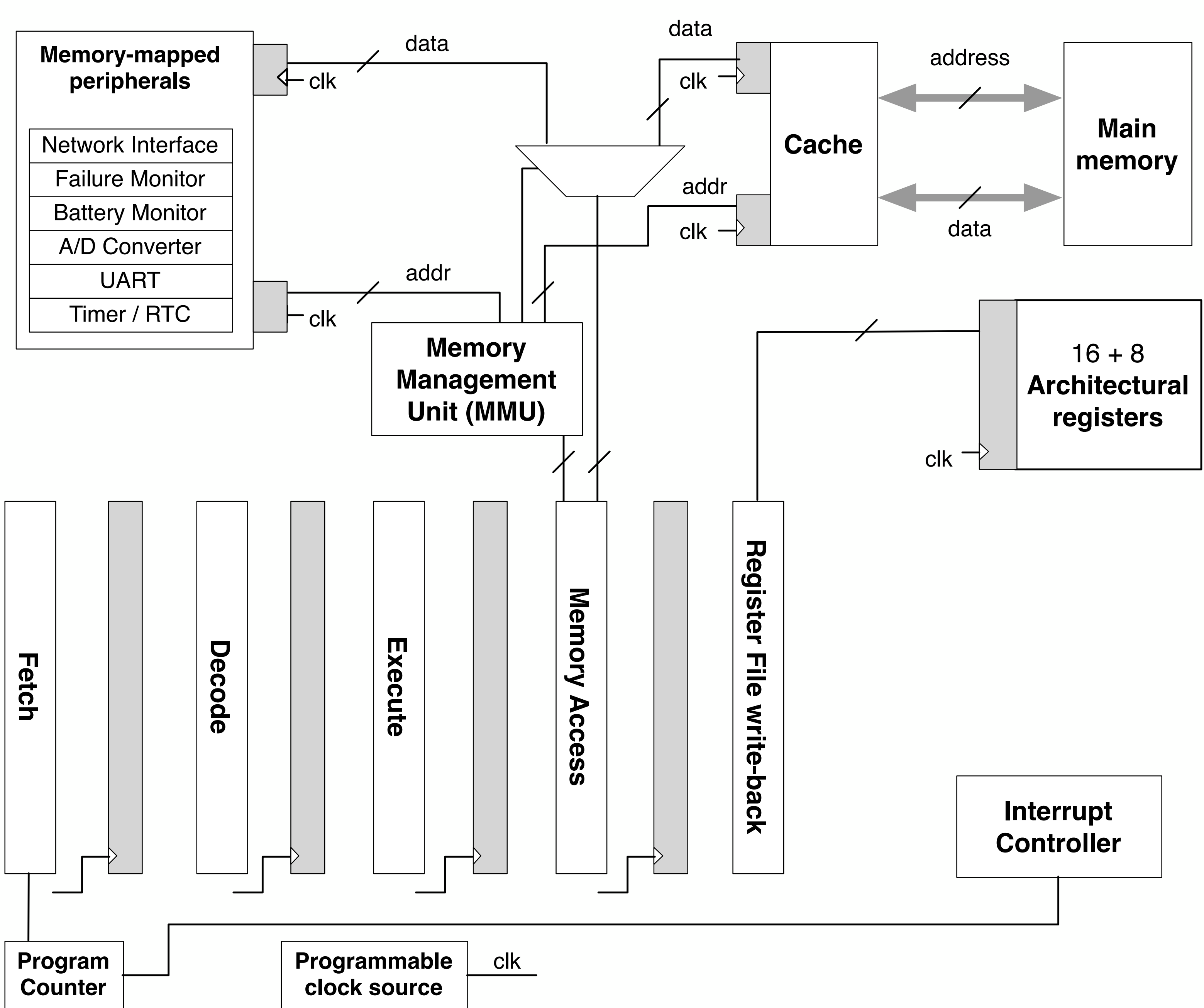
```
node ID=0, PC=0x80042ba, ICLK=1668327, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],2
EX: []
ID: []
IF: [0xd109],0
```

```
node ID=0, PC=0x80042bc, ICLK=1668328, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],1
EX: []
ID: [0xd109],0
IF: [0x6413],0
```

```
node ID=0, PC=0x80042be, ICLK=1668329, sleep?=0
buslock=1, buslocker=0, EX touches mem = 1
WB: [ANDI],0
MA: []
EX: [MOVBSG],1
ID: [0x6413],0
IF: [0xd109],0
```

```
node ID=0, PC=0x80042c0, ICLK=1668330, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [MOVBSG],0
EX: [MACL],1
ID: [0xd109],0
IF: [0x410b],0
```

...



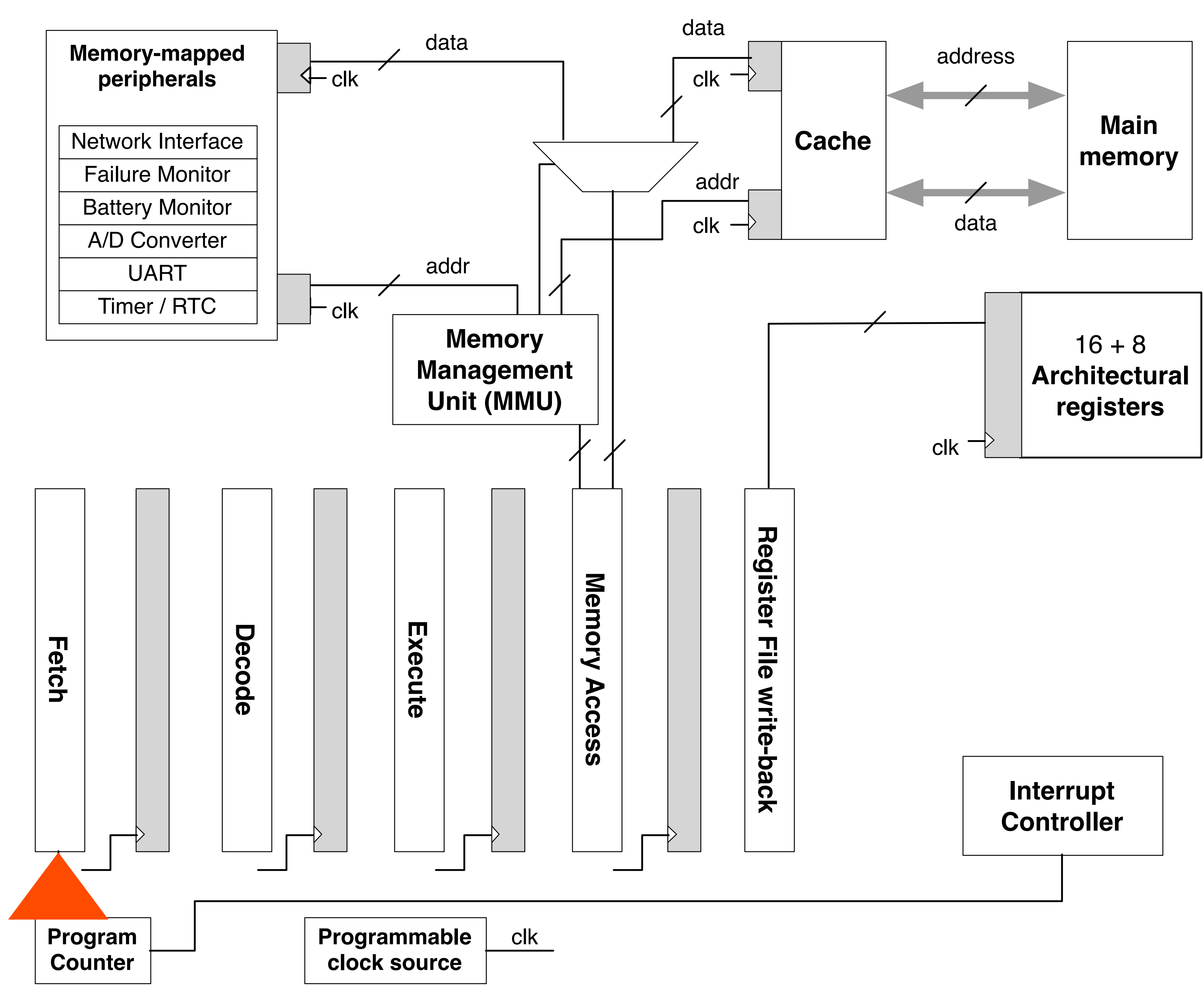
```
node ID=0, PC=0x80042ba, ICLK=1668327, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],2
EX: []
ID: []
IF: [0xd109],0
```

```
node ID=0, PC=0x80042bc, ICLK=1668328, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],1
EX: []
ID: [0xd109],0
IF: [0x6413],0
```

```
node ID=0, PC=0x80042be, ICLK=1668329, sleep?=0
buslock=1, buslocker=0, EX touches mem = 1
WB: [ANDI],0
MA: []
EX: [MOVBSG],1
ID: [0x6413],0
IF: [0xd109],0
```

```
node ID=0, PC=0x80042c0, ICLK=1668330, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [MOVBSG],0
EX: [MACL],1
ID: [0xd109],0
IF: [0x410b],0
```

...



12

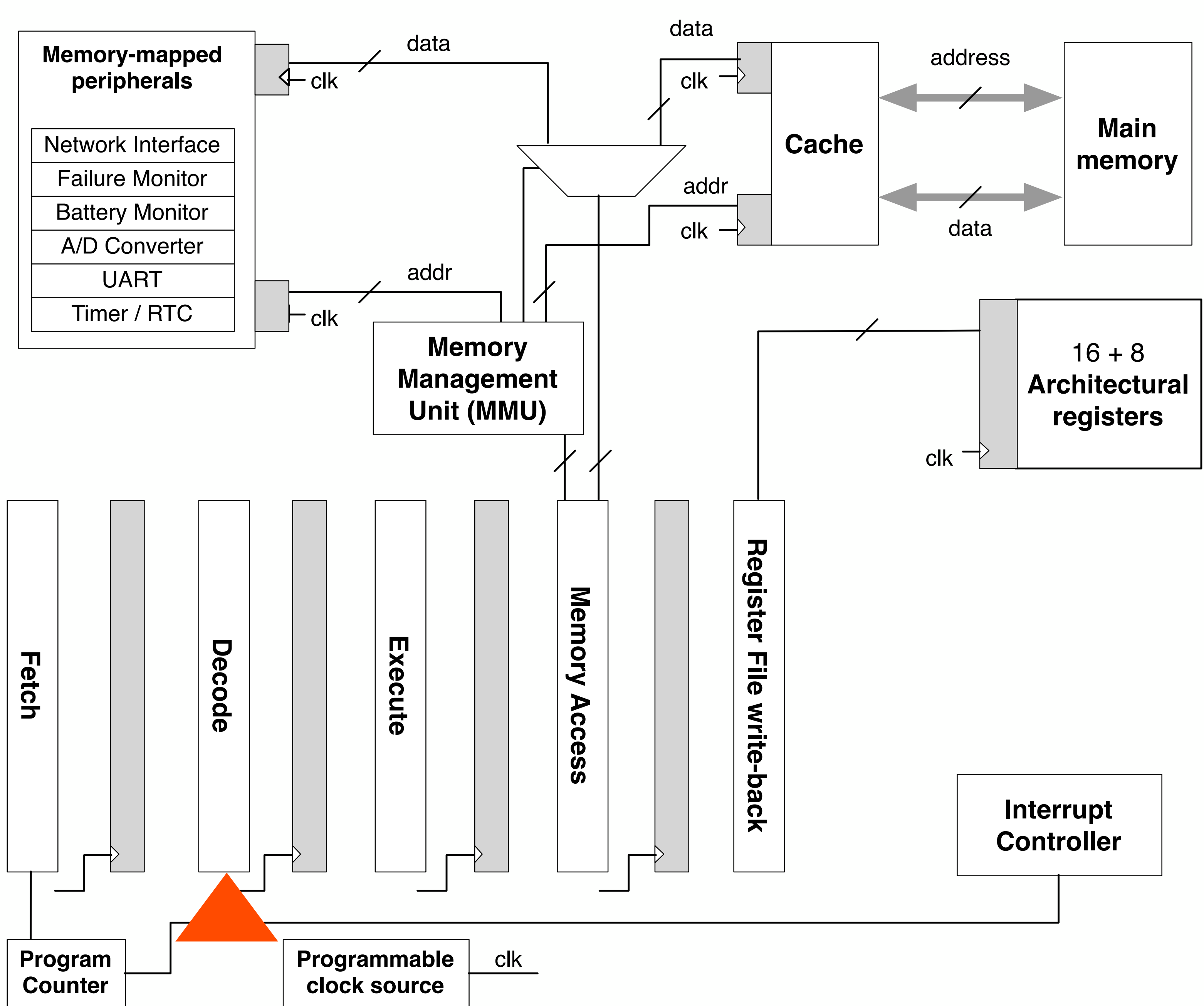
node ID=0, PC=0x80042ba, ICLK=1668327, sleep?=0  
buslock=1, buslocker=0, EX touches mem = 0  
WB: ☐  
MA: [ANDI],2  
EX: ☐  
ID: ☐  
IF: [0xd109],0

node ID=0, PC=0x80042bc, ICLK=1668328, sleep?=0  
buslock=1, buslocker=0, EX touches mem = 0  
WB: ☐  
MA: [ANDI],1  
EX: ☐  
ID: [0xd109],0  
IF: [0x6413],0

node ID=0, PC=0x80042be, ICLK=1668329, sleep?=0  
buslock=1, buslocker=0, EX touches mem = 1  
WB: [ANDI],0  
MA: ☐  
EX: [MOVBSG],1  
ID: [0x6413],0  
IF: [0xd109],0

node ID=0, PC=0x80042c0, ICLK=1668330, sleep?=0  
buslock=1, buslocker=0, EX touches mem = 0  
WB: ☐  
MA: [MOVBSG],0  
EX: [MACL],1  
ID: [0xd109],0  
IF: [0x410b],0

...



```
node ID=0, PC=0x80042ba, ICLK=1668327, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],2
EX: []
ID: []
IF: [0xd109],0
```

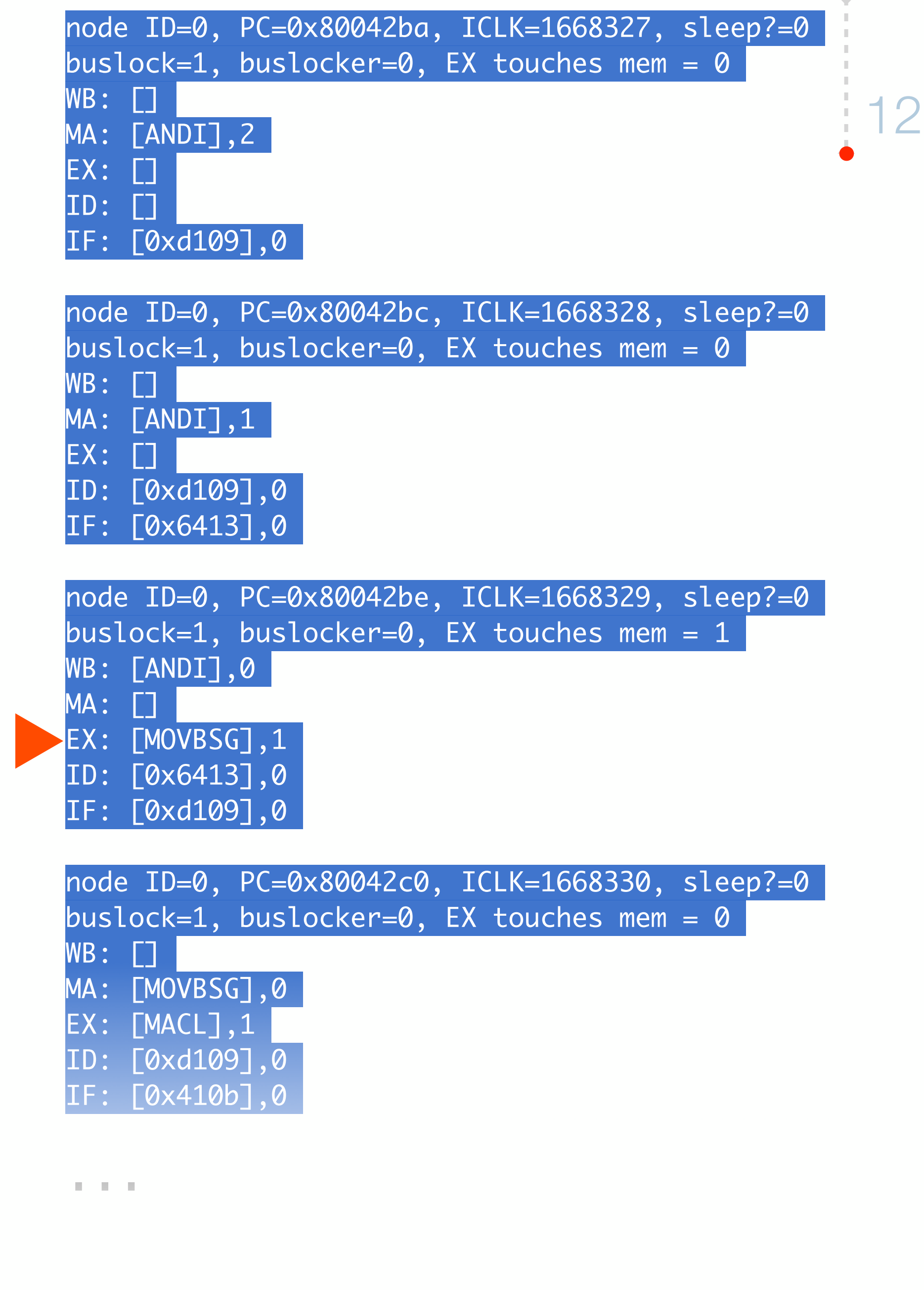
```
node ID=0, PC=0x80042bc, ICLK=1668328, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],1
EX: []
ID: [0xd109],0
IF: [0x6413],0
```

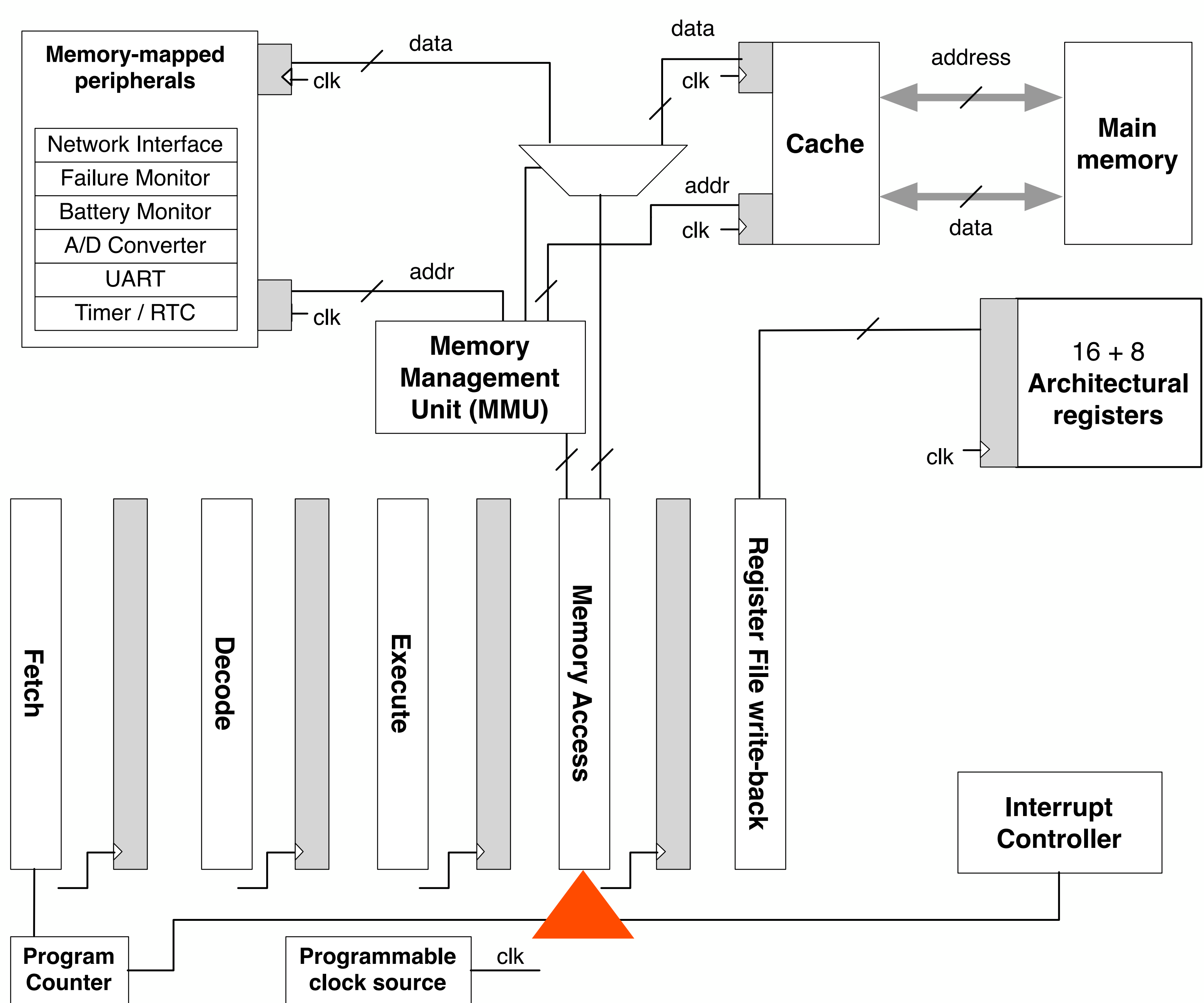
```
node ID=0, PC=0x80042be, ICLK=1668329, sleep?=0
buslock=1, buslocker=0, EX touches mem = 1
WB: [ANDI],0
MA: []
EX: [MOVBSG],1
ID: [0x6413],0
IF: [0xd109],0
```

```
node ID=0, PC=0x80042c0, ICLK=1668330, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [MOVBSG],0
EX: [MACL],1
ID: [0xd109],0
IF: [0x410b],0
```

...







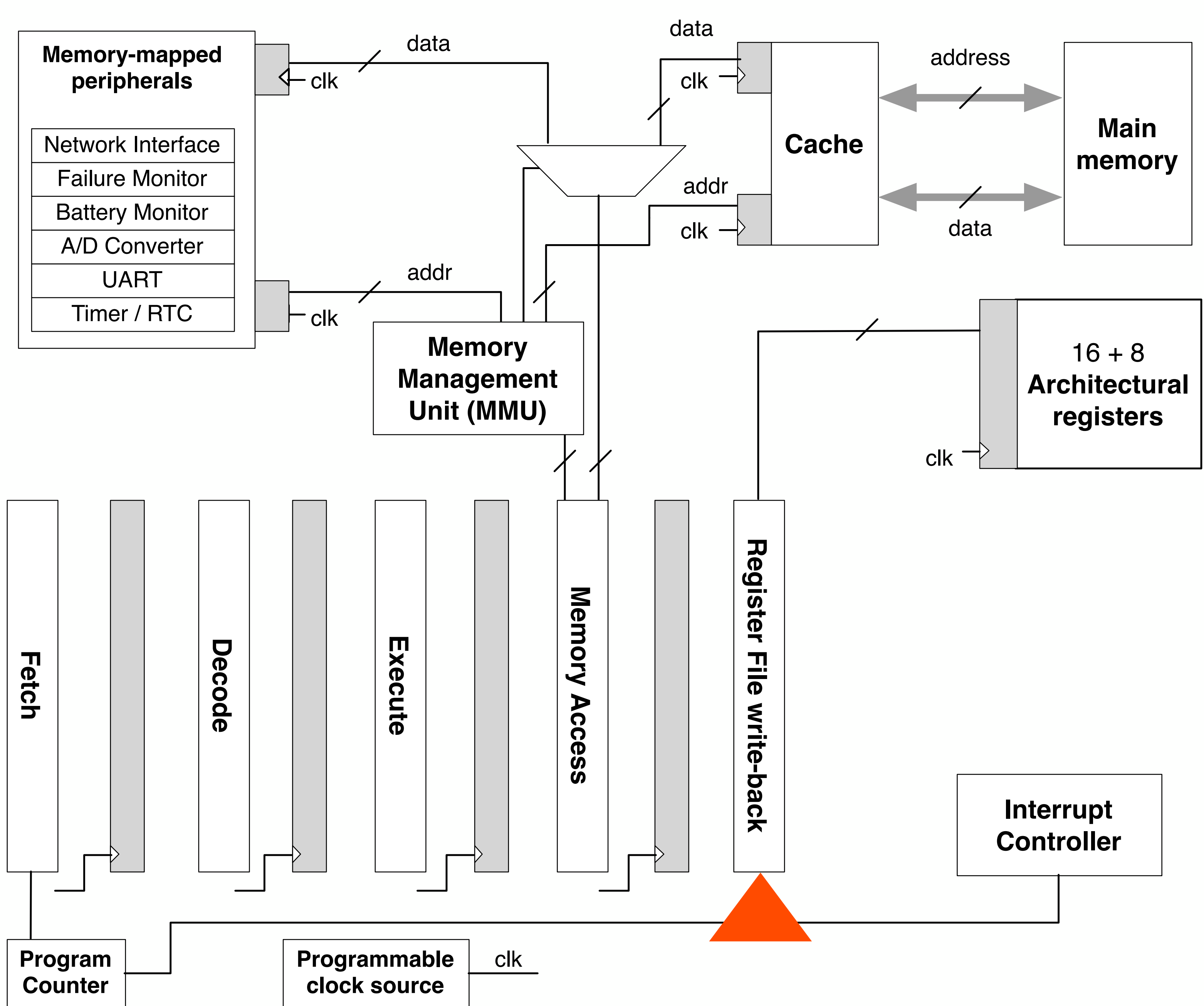
```
node ID=0, PC=0x80042ba, ICLK=1668327, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],2
EX: []
ID: []
IF: [0xd109],0
```

```
node ID=0, PC=0x80042bc, ICLK=1668328, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],1
EX: []
ID: [0xd109],0
IF: [0x6413],0
```

```
node ID=0, PC=0x80042be, ICLK=1668329, sleep?=0
buslock=1, buslocker=0, EX touches mem = 1
WB: [ANDI],0
MA: []
EX: [MOVBSG],1
ID: [0x6413],0
IF: [0xd109],0
```

```
node ID=0, PC=0x80042c0, ICLK=1668330, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [MOVBSG],0
EX: [MACL],1
ID: [0xd109],0
IF: [0x410b],0
```

...



```
node ID=0, PC=0x80042ba, ICLK=1668327, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],2
EX: []
ID: []
IF: [0xd109],0
```

```
node ID=0, PC=0x80042bc, ICLK=1668328, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [ANDI],1
EX: []
ID: [0xd109],0
IF: [0x6413],0
```

```
node ID=0, PC=0x80042be, ICLK=1668329, sleep?=0
buslock=1, buslocker=0, EX touches mem = 1
WB: [ANDI],0
MA: []
EX: [MOVBSG],1
ID: [0x6413],0
IF: [0xd109],0
```

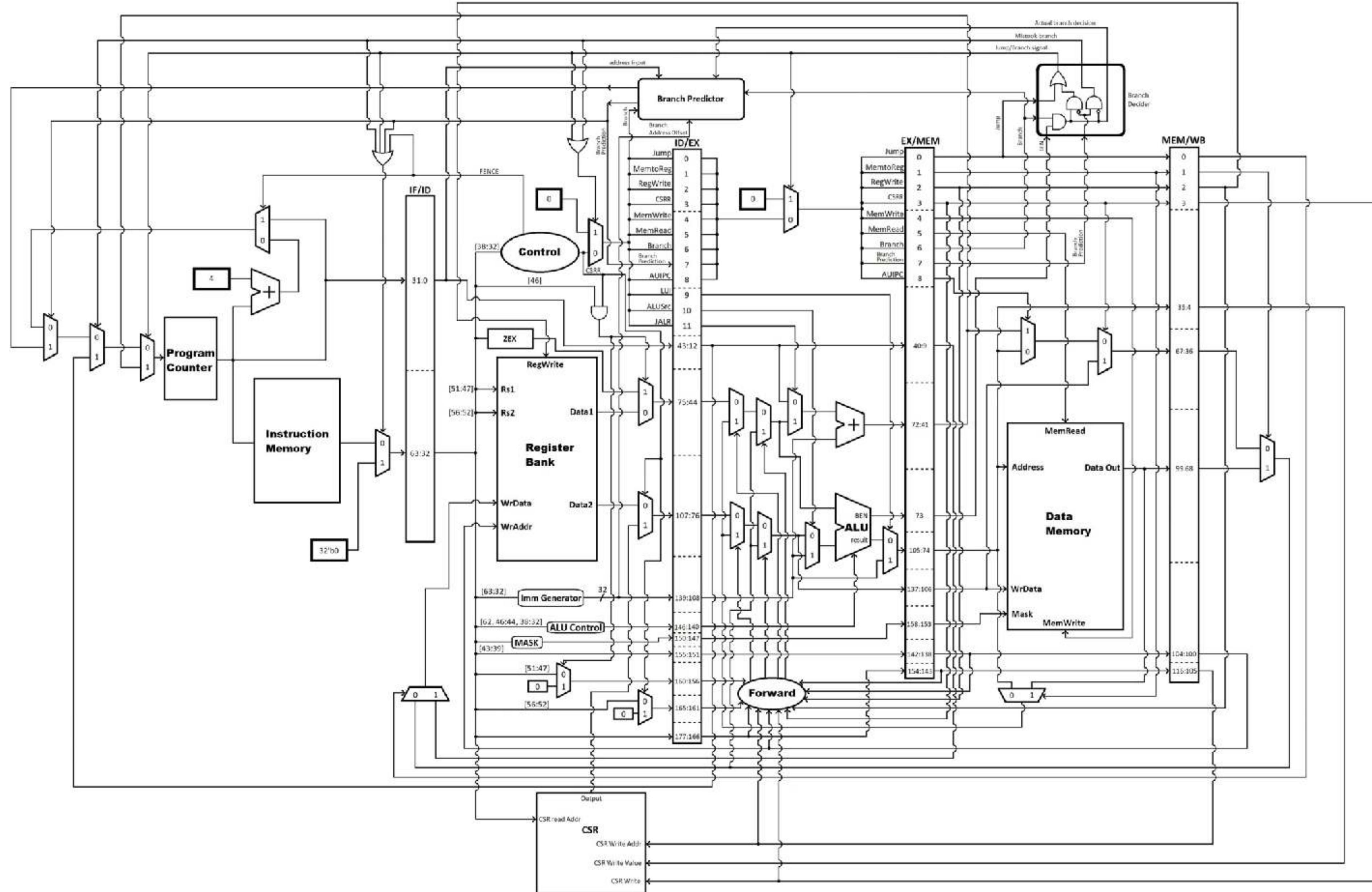
```
node ID=0, PC=0x80042c0, ICLK=1668330, sleep?=0
buslock=1, buslocker=0, EX touches mem = 0
WB: []
MA: [MOVBSG],0
EX: [MACL],1
ID: [0xd109],0
IF: [0x410b],0
```

...

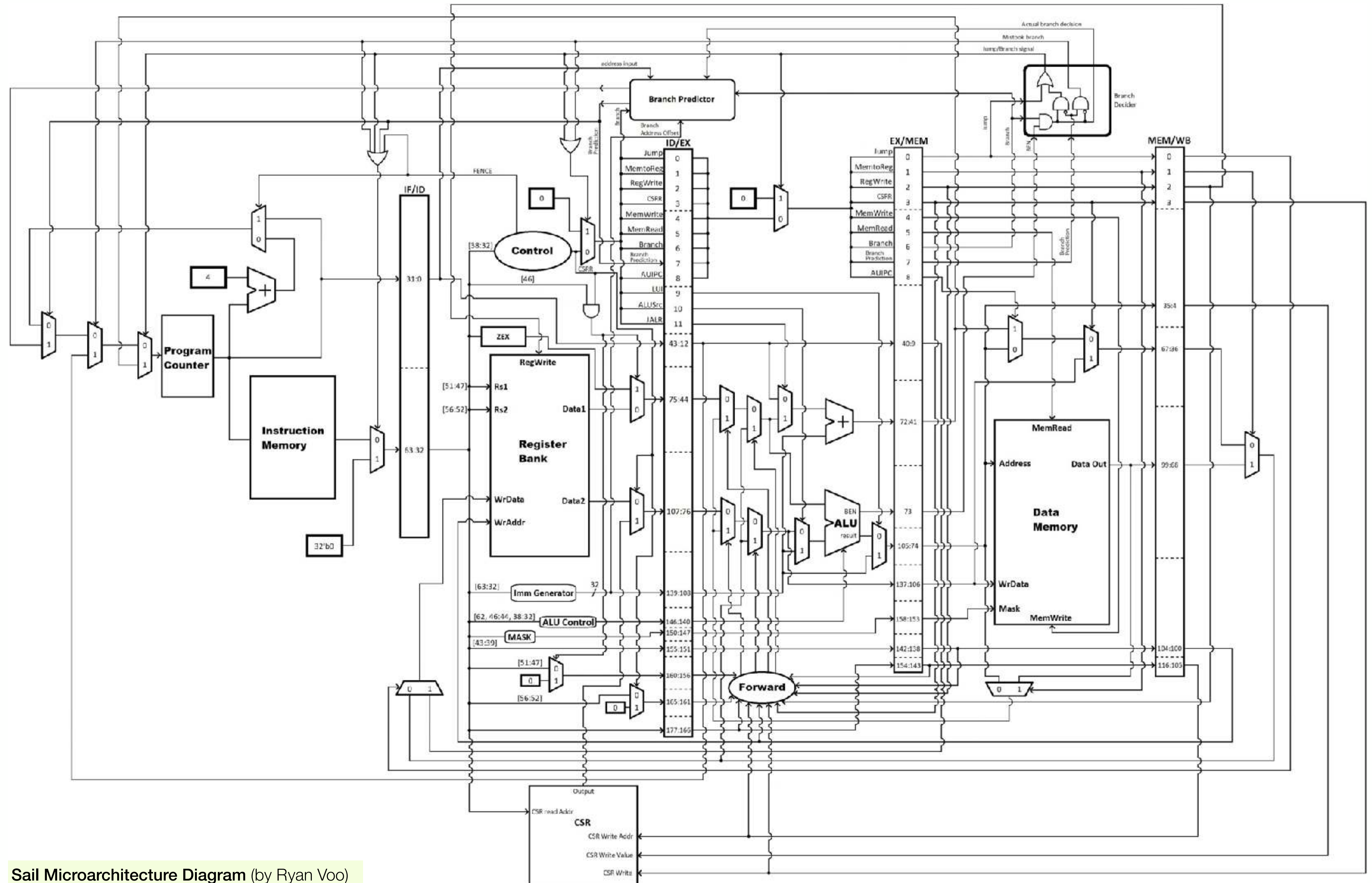
# Example: Sail 5-Stage Pipeline

8

12

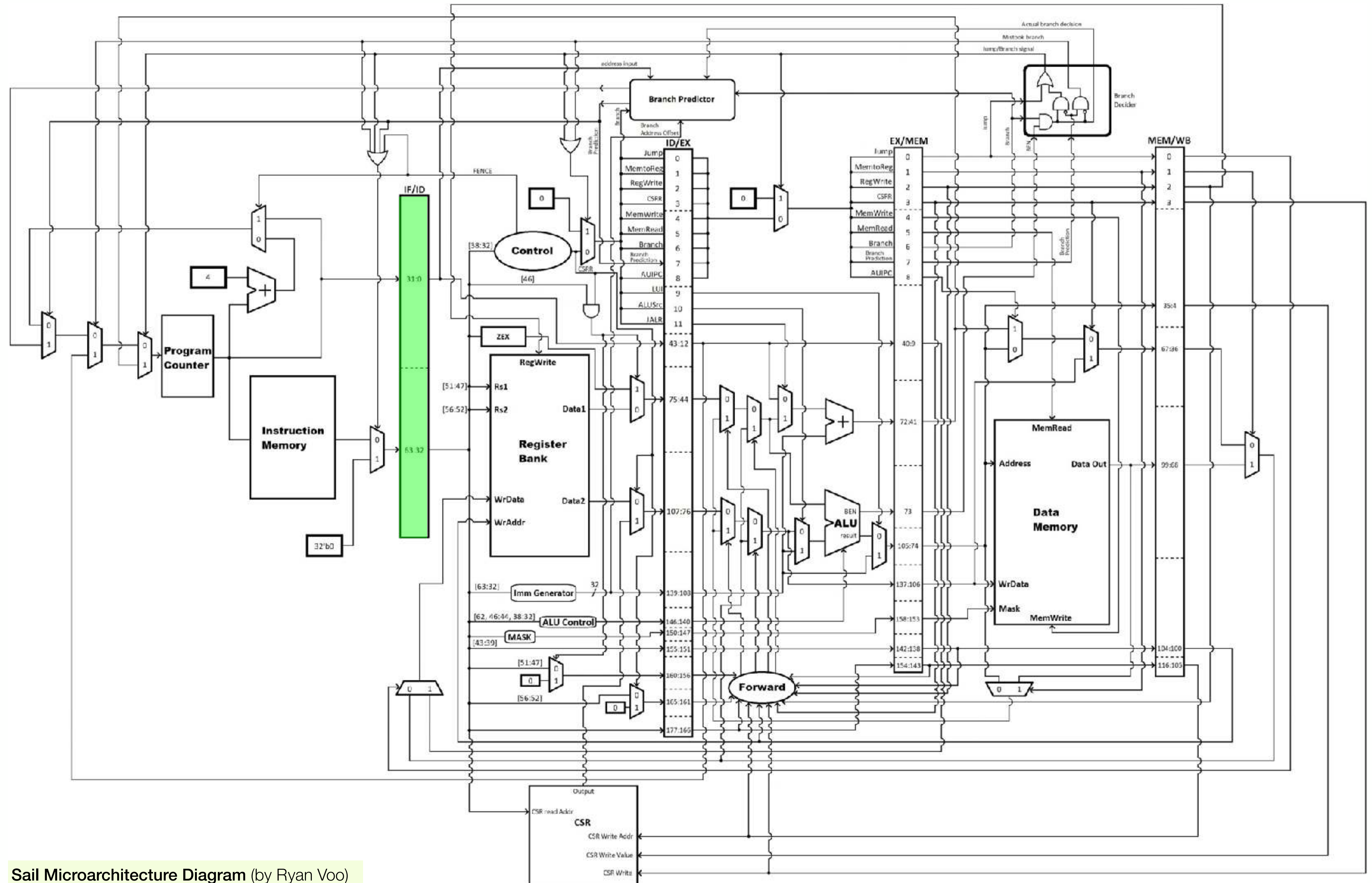




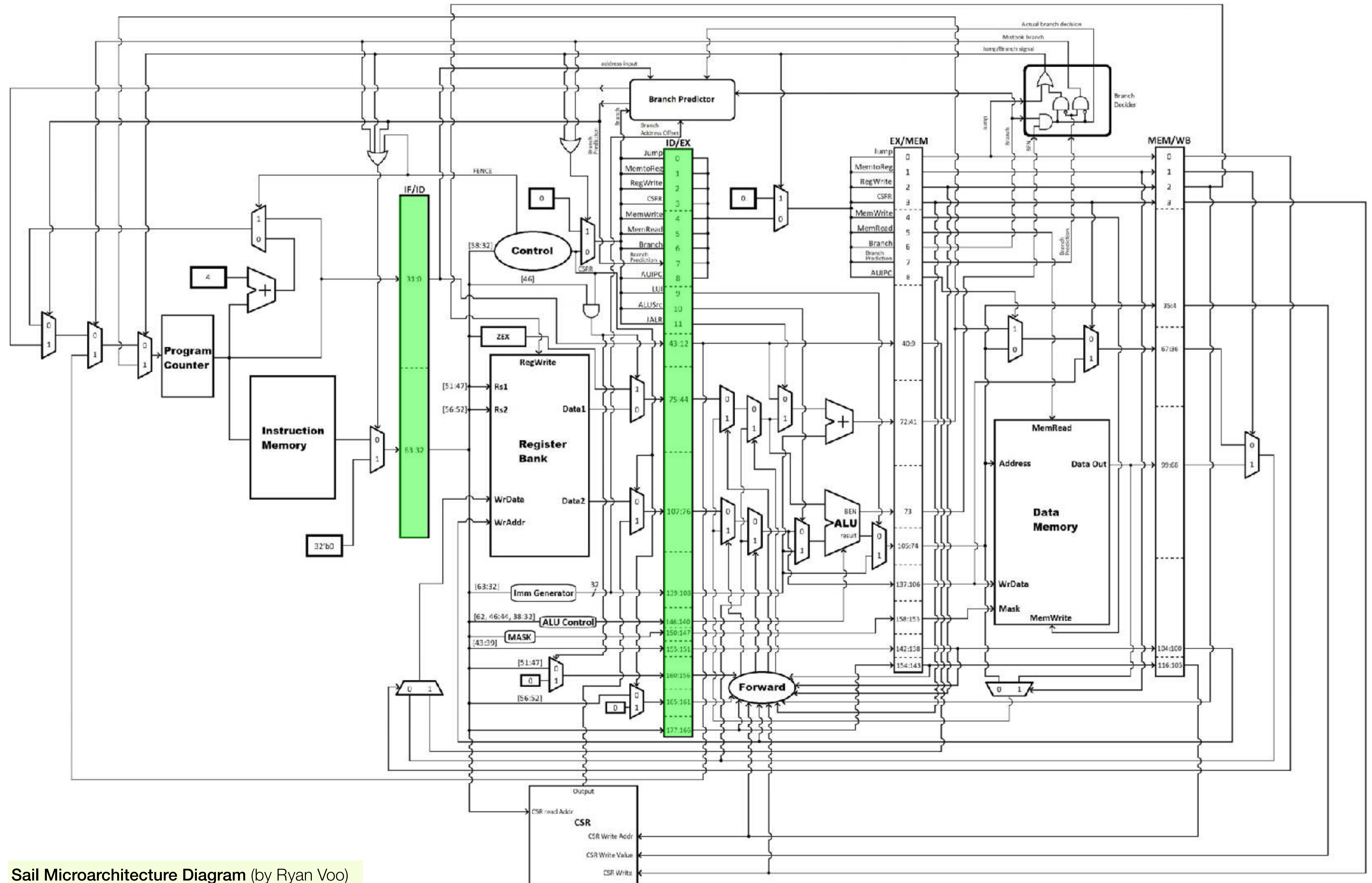


Sail Microarchitecture Diagram (by Ryan Voo)



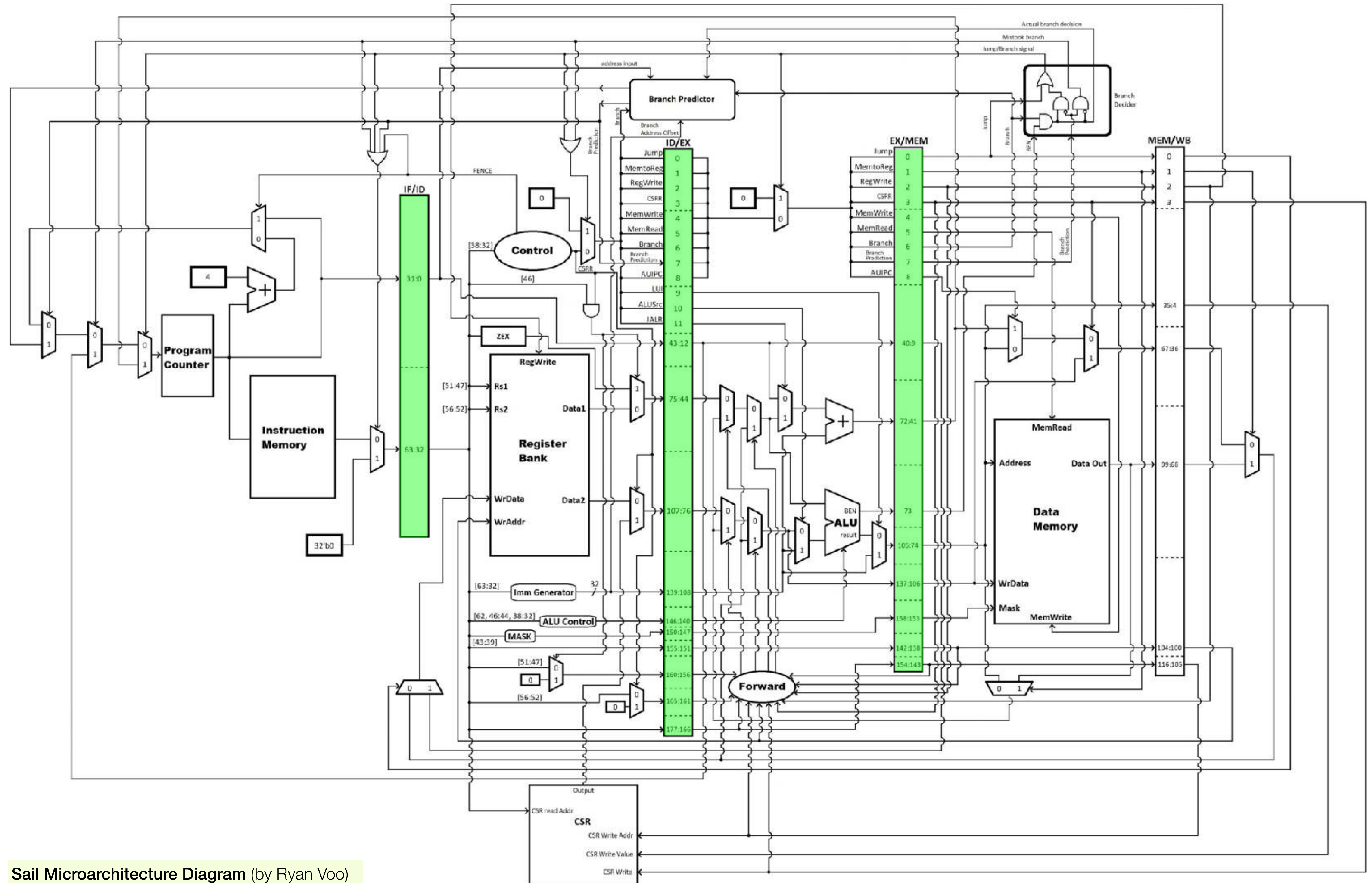


Sail Microarchitecture Diagram (by Ryan Voo)



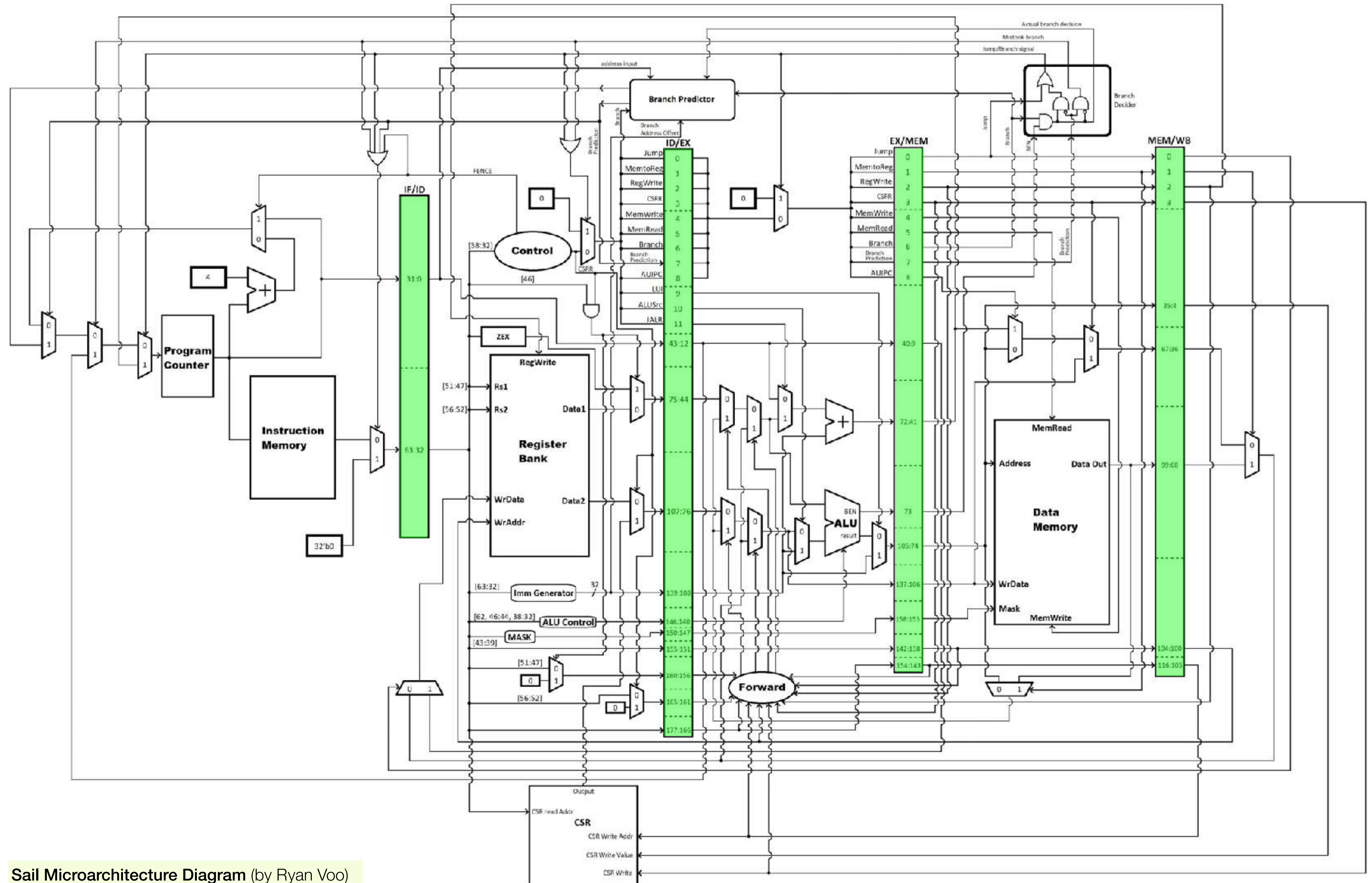
Sail Microarchitecture Diagram (by Ryan Voo)





Sail Microarchitecture Diagram (by Ryan Voo)





Sail Microarchitecture Diagram (by Ryan Voo)



# Pipelining Improves Throughput

# Pipelining Improves Throughput

The time taken for each individual instruction to complete remains the same

# Pipelining Improves Throughput

The time taken for each individual instruction to complete remains the same

Multiple instructions are in the pipeline at a given time—pipeline parallelism

# Instruction Sets and Pipelining

---

11

12

**What we want for efficient pipelining:**

# Instruction Sets and Pipelining

---

11

12

## **What we want for efficient pipelining:**

- ▶ Stages that have similar complexity

# Instruction Sets and Pipelining

---

11

12

## **What we want for efficient pipelining:**

- ▶ Stages that have similar complexity
- ▶ Instructions that are easy to fetch: short instructions, fixed length



## **What we want for efficient pipelining:**

- ▶ Stages that have similar complexity
- ▶ Instructions that are easy to fetch: short instructions, fixed length
- ▶ Instructions that are easy to decode (small number of formats, simple fields)

## **What we want for efficient pipelining:**

- ▶ Stages that have similar complexity
- ▶ Instructions that are easy to fetch: short instructions, fixed length
- ▶ Instructions that are easy to decode (small number of formats, simple fields)
- ▶ Simple instruction operations (e.g., no combined arithmetic and memory access)

## **What we want for efficient pipelining:**

- ▶ Stages that have similar complexity
- ▶ Instructions that are easy to fetch: short instructions, fixed length
- ▶ Instructions that are easy to decode (small number of formats, simple fields)
- ▶ Simple instruction operations (e.g., no combined arithmetic and memory access)

**RISC-V was designed to make it easier to achieve good pipelining**

## **What we want for efficient pipelining:**

- ▶ Stages that have similar complexity
- ▶ Instructions that are easy to fetch: short instructions, fixed length
- ▶ Instructions that are easy to decode (small number of formats, simple fields)
- ▶ Simple instruction operations (e.g., no combined arithmetic and memory access)

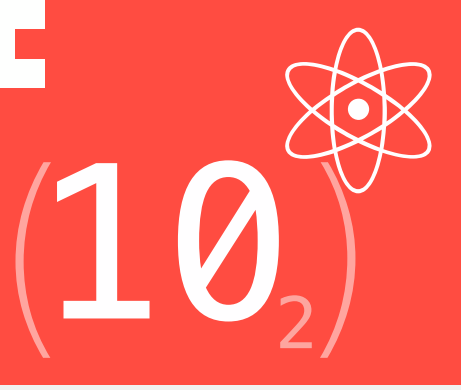
## **RISC-V was designed to make it easier to achieve good pipelining**

- ▶ Contrast with ARM and Intel x86

# Things to Do

---

- ❶ Drop off a **“muddiest point” sheet when you can**



University of Cambridge  
Department of Engineering

**Physical  
Computation  
Laboratory**