27.07.20 12:40

CSCI 150
Introduction to Digital and Computer System Design
Lecture 5: Registers I

Jetic Gū

2020 Summer Semester (S2)

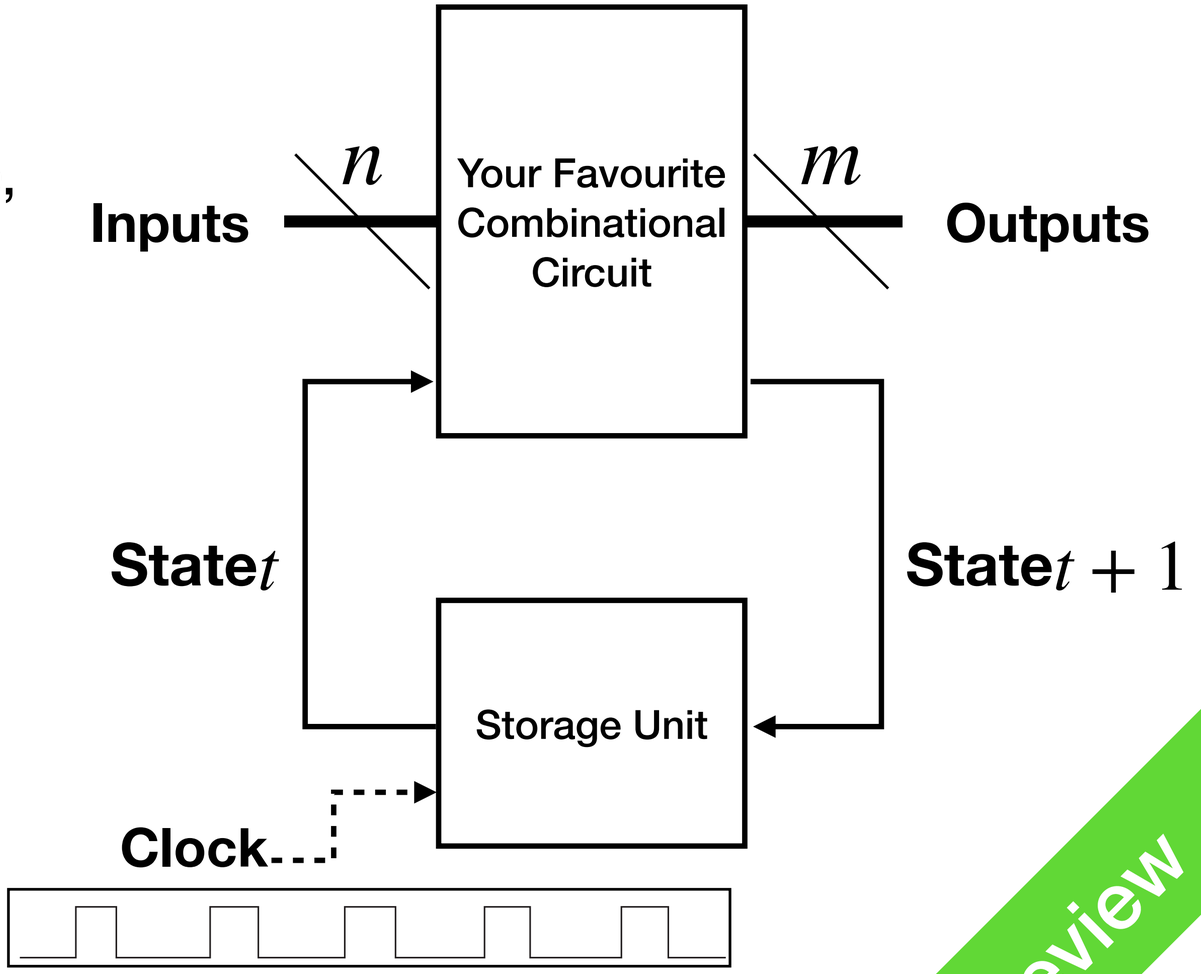
Overview

- Focus: Fundamentals of Complex Digital Circuit Design
- Architecture: von Neumann
- Textbook v4: Ch7 7.1 7.2; v5: Ch6 6.1 6.2
- Core Ideas:
    1. What are Registers
    2. Register Transferring Operations and Circuit

P0
Review

Sequential Circuits

- **Synchronous Sequential Circuit**
  Signals arrive at discrete instants of time, outputs at next time step
  - Has Clock
- **Asynchronous Sequential Circuit**
  Signals arrive at any instant of time, outputs when ready
  - May not have Clock



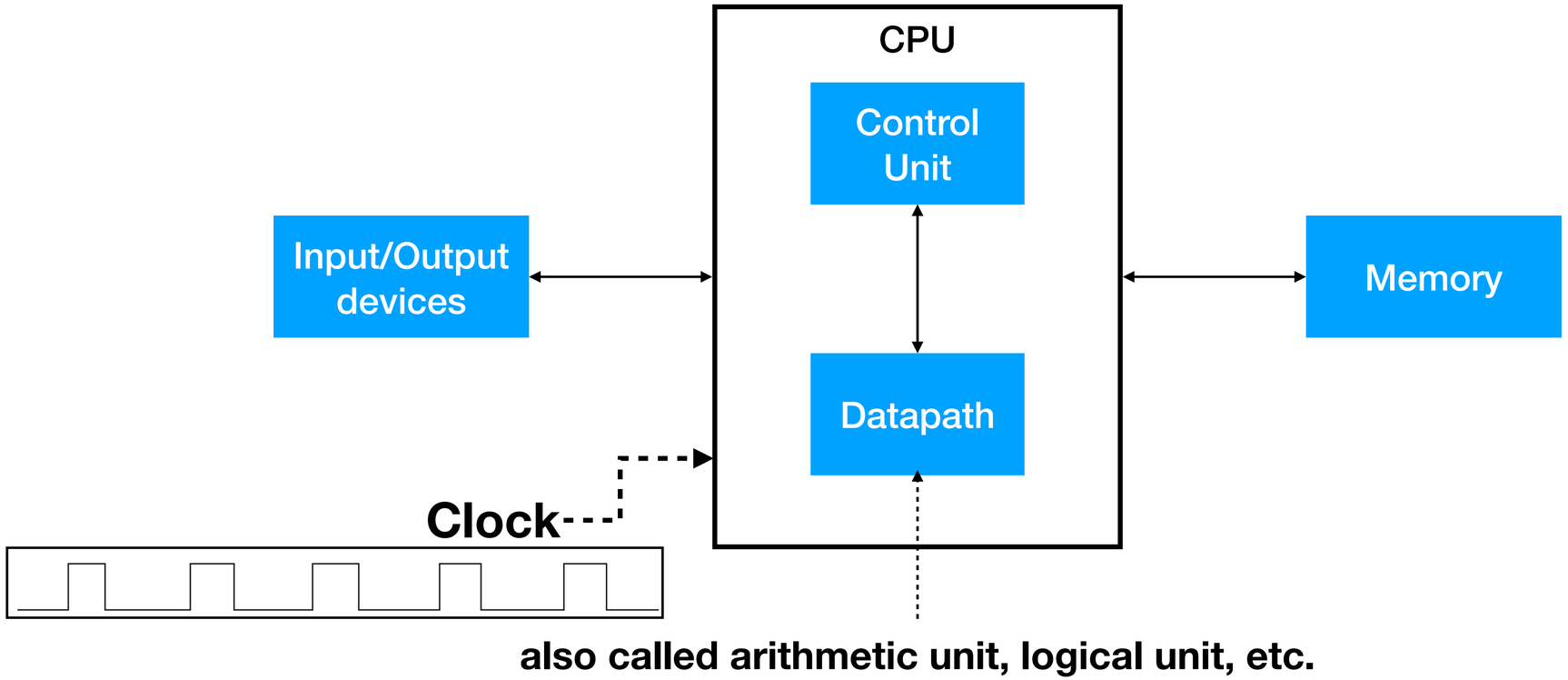
Review

What are Registers?

Definitions;
Register Loading; Parallel Loading

Summary

Computer



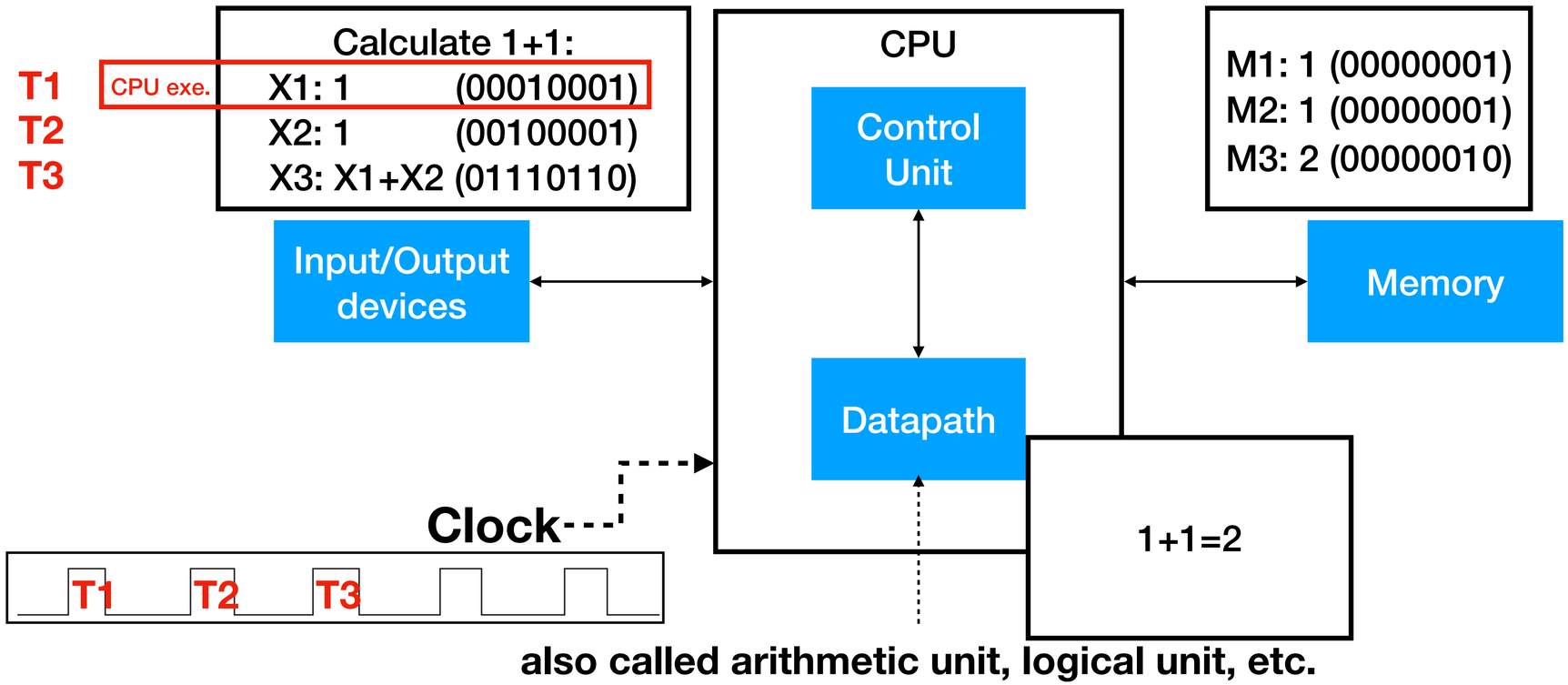
also called arithmetic unit, logical unit, etc.

1. Von Neumann Architecture

Review

Computer

A very rough example



1. Von Neumann Architecture

Review

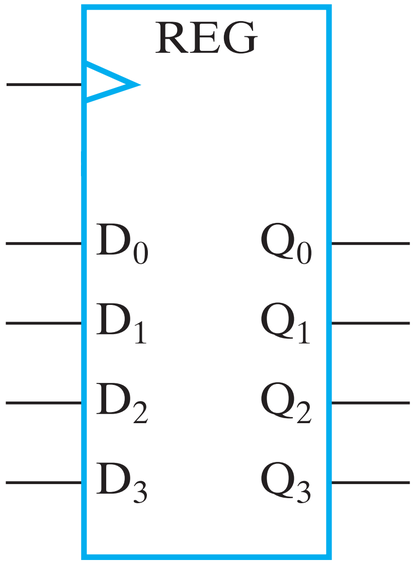
von Neumann CPU

- Control Unit
  - Determine sequence of data-processing operations performed by the datapath
- Datapath
  - Processing logic units: **Adder**, **Subtractor**, **Shifter**, **Counter**, etc.
  - Registers: **Storage** of temporary information, **basic components** of the digital system

Concept

Register

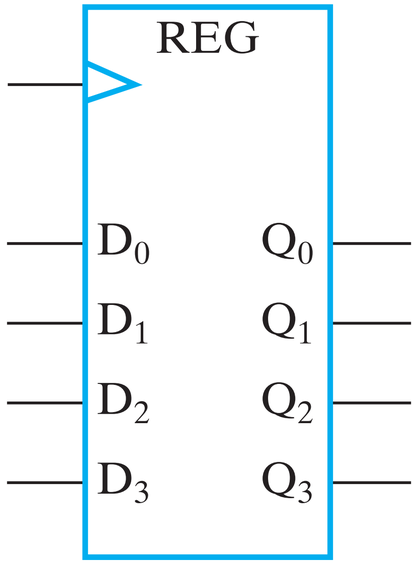
- n-bit register: uses n flip-flops
  stores n bits of information

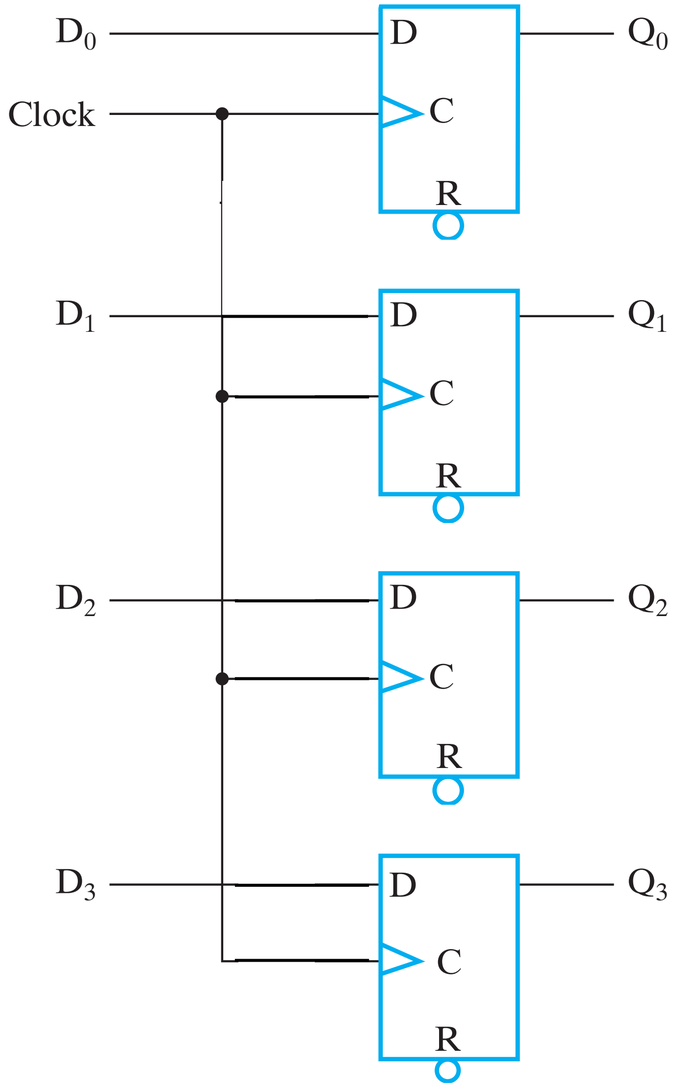


Concept

Register

- n-bit register: uses n flip-flops stores n bits of information
- An array of D flip-flops with reset

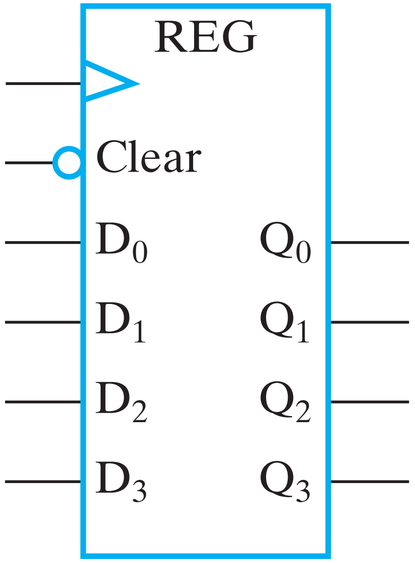


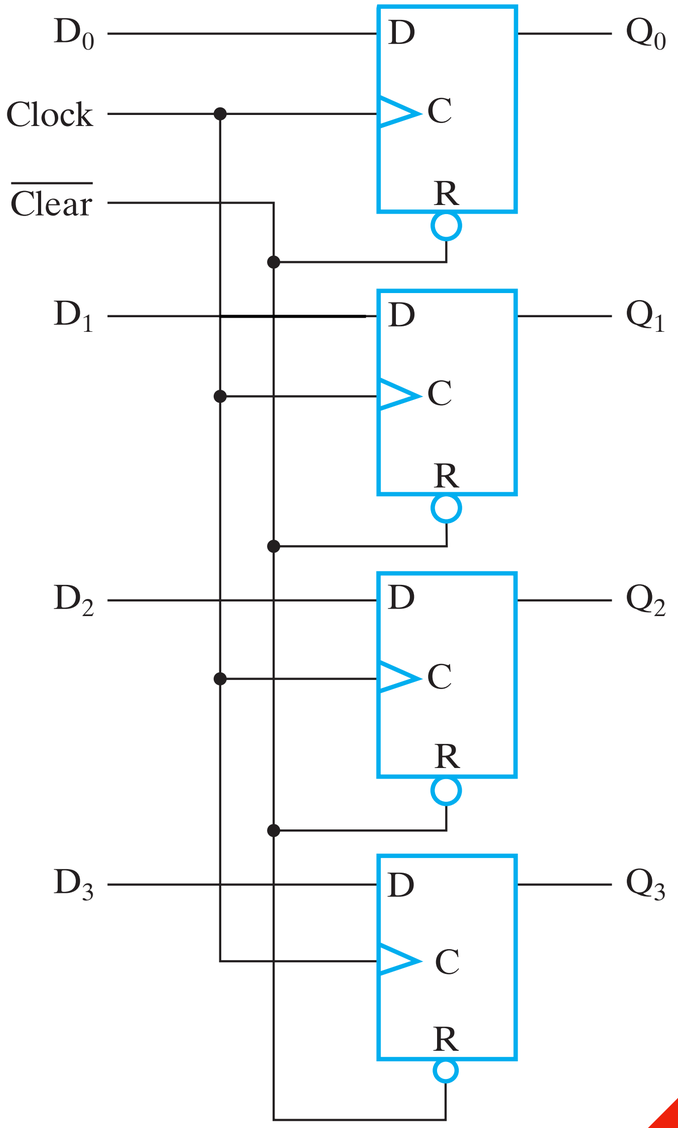


Concept

Register

- n-bit register: uses n flip-flops
  stores n bits of information
- An array of D flip-flops with reset
- `Clear`: set register to all 0s

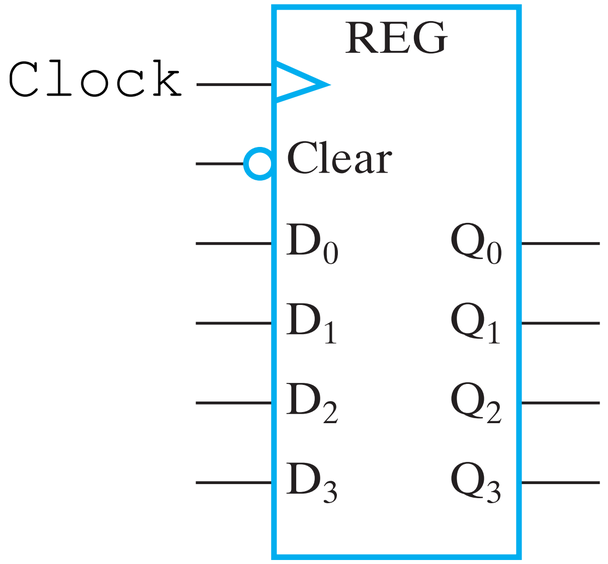




Concept

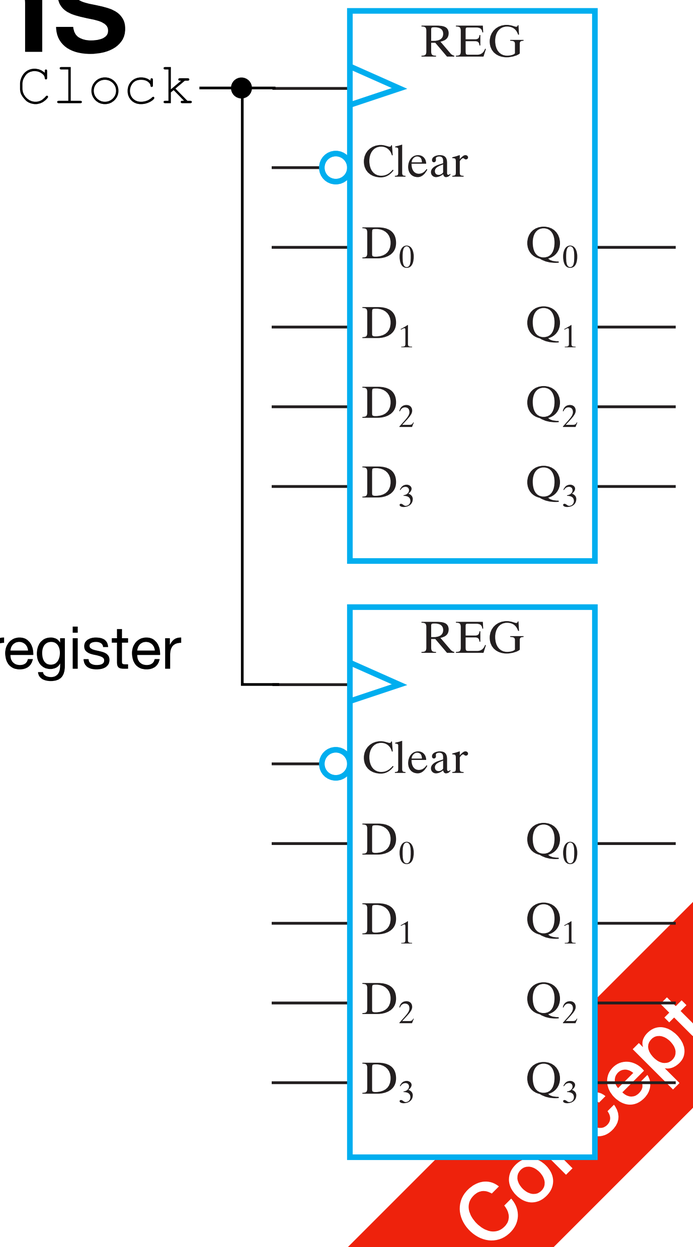
Register

- n-bit register: uses n flip-flops
  stores n bits of information
- An array of D flip-flops with reset
- `Clear`: set register to all 0s
- `Loading`: set register to $D_{3:0}$
  Triggered by `Load` or `Clock`



Concept

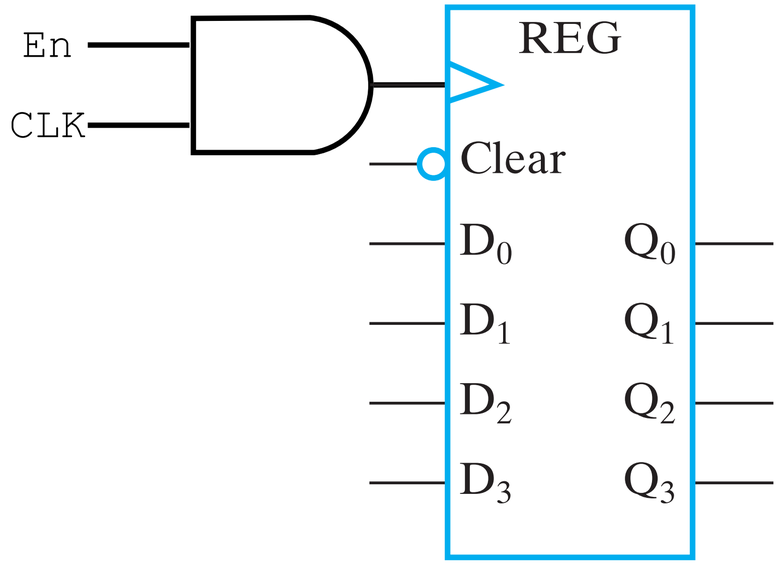
Register Operations



- All registers are most likely wired to one `Clock`
- `Loading` a register: assigning new values to all n-bits of a register
- `Clearing` a register: change all n-bits of a register to 0s

What if we don't want to change the value of a register?

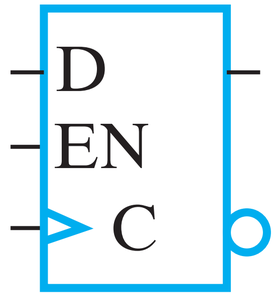
- `Clock`: generates a constant train of pulses triggering the `C` of each registers
- **Clock gating**
  - Adding an Enabler to each `C` of each register
  - Bad idea: leads to different propagation delay between the `CLK` and the Input D



Concept

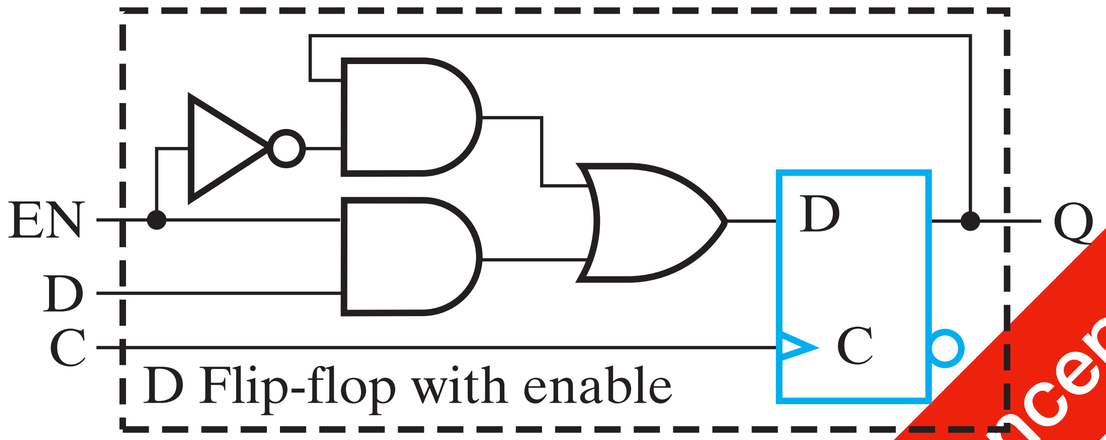
What if we don't want to change the value of a register?

- `Clock`: generates a constant train of pulses triggering the `C` of each registers



- Use D flip-flops with **built-in Enabler (Correct!)**
  - The `CLK` goes directly to C
  - Input D combined with EN signal
    Ensure same propagation delay design

Why can't we use a regular enabler at D?



Concept

Registers

- What is a register?
- Basic Functions of a single Register
  - `Loading`: set values to input
  - `Clearing`: set values to 0
  - `Enabling`: preserving existing values

Review

P2
Example

x86 Registers

And how addition is performed on your Computer CPU

Summary

P2
Example

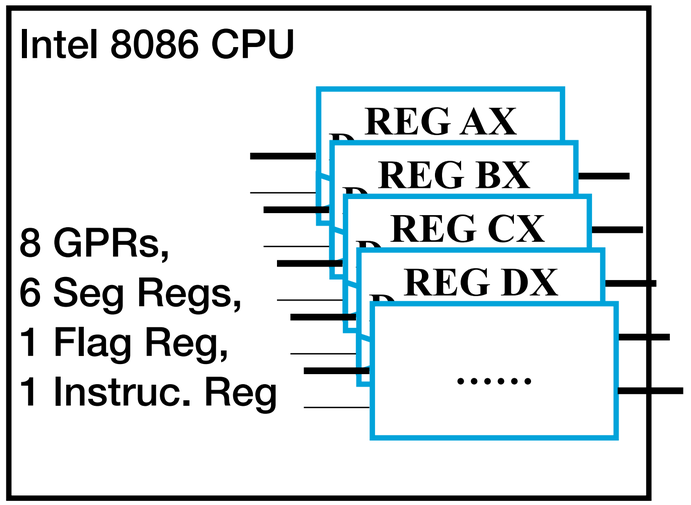
Common CPU Processor Architectures

- These are all von Neumann architecture designs
  - X86 architecture (Intel CPUs, AMD CPUs)
  - X86-64 architecture (64bit version of X86)
  - ARM (iPhone, iPad, most Android devices)
  - MIPS (Others, including instructional)

Concept

P2
Example

X86 CPU Registers



- These registers are located on the CPU chips (in Datapath)
- 8 General-Purpose Registers (GPRs)
  - **AX**: Accumulator register. Used in arithmetic operations
  - **BX**: Base register. Used as a pointer to data
  - **CX**: Counter register. Used in shift/rotate instructions and loops.
  - **DX**: Data register. Used in arithmetic operations and I/O operations.
  - ……

Concept

P2
Example

X86 Addition

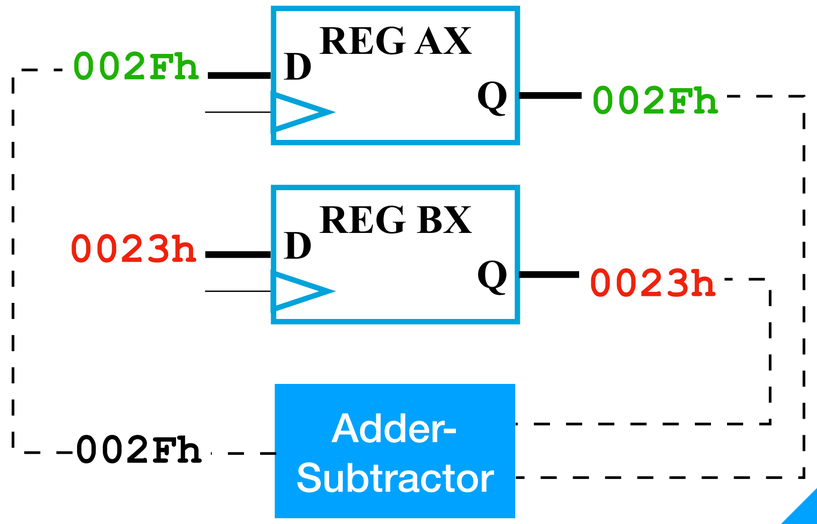
- 12 + 35
- Uses AX, BX: `Clear`ed to 0

1. **Load** AX with 12 (`000Ch`)
2. **Load** BX with 35 (`0023h`)
3. Perform **Add** with Adder-Subtractor, **Load** result to AX

Assembly Language

```
MOV AX, 000Ch
MOV BX, 0023h
ADD AX, BX
```

Hardware



Demo

P2
Example

Datapath and Control Unit

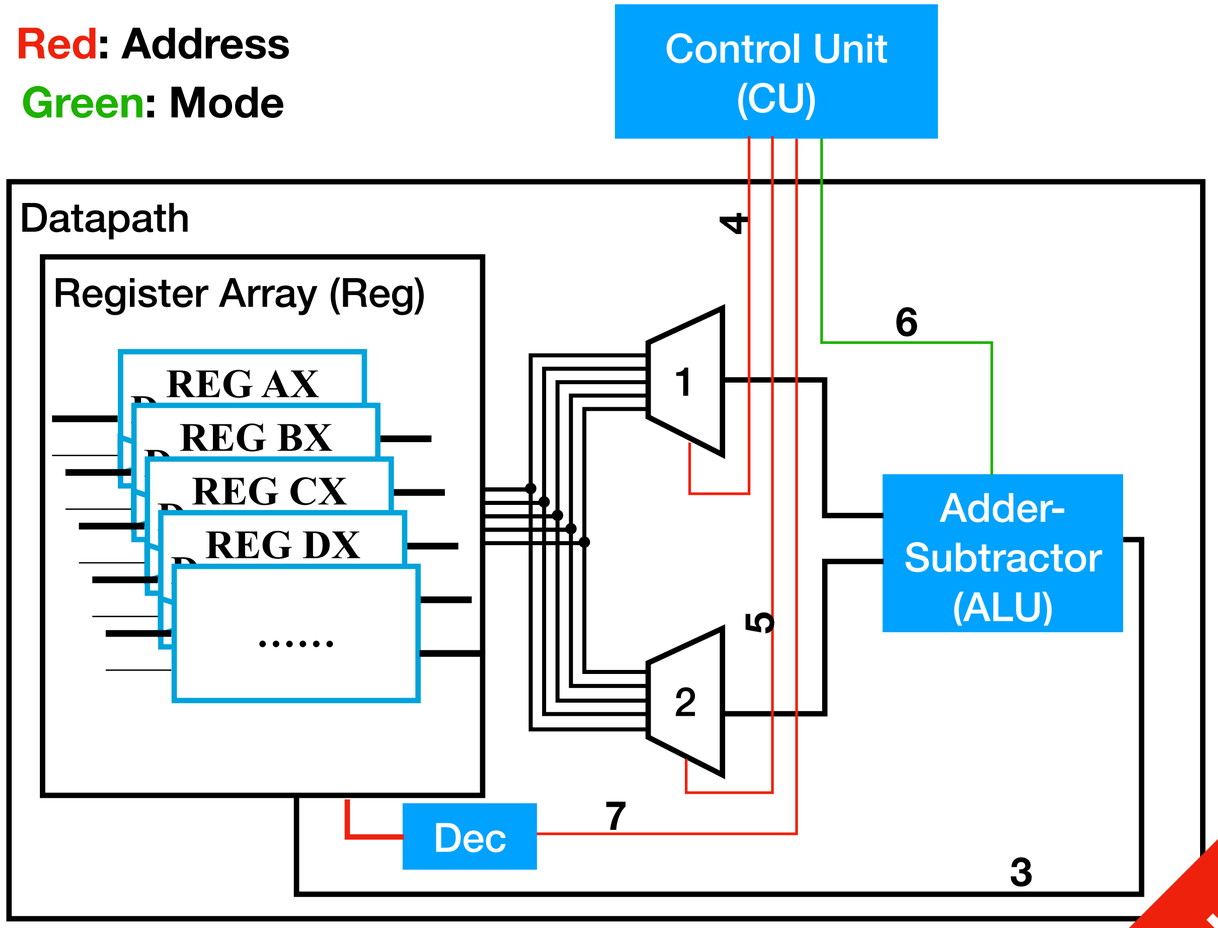
- Control Unit at each time step, provide
  - Operation Code
    e.g. `mov` (`66b8`, etc.), `add` (`6601`, etc.)
  - Parameters
    e.g. `ax`, `bx`, 12, 35
- Datapath
  - Select Register for Input and Output (Multiplexer)
  - Feed input into Register or Functional Blocks (Adder-Subtractor)

Concept

P2
Example

Datapath and Control Unit

1. `ALU` connected to `Reg` for 1st input
2. `ALU` connected to `Reg` for 2nd input
3. `ALU` connected to `Reg` to store result
4. `CU` tells `ALU` which register to take as 1st input
5. `CU` tells `ALU` which register to take as 2st input
6. `CU` tells `ALU` which operation to do
7. `CU` tells which `Reg` to store result in
   Using decoder and `EN` on each Register



Concept

Register Transferring

Microoperations; Transferring Operations

Summary

Register Operations

- Movement of data stored in registers and Processing performed on the data
- Components
  - set of registers in the system
  - operations performed on the data
  - control that supervises the sequence of operations in the system

Concept

Microoperation

- **Microoperation**: An elementary operation performed on data stored in registers
  - Single Register (**Transfer Operations**): `load`, `clear`, `shift`, `count`, etc.
  - Multiple Registers: `add`, `subtract`, etc.

Concept

Register Transfer VHDL

	Operator	Example
Assignment	<=	ax **<=** 12h
Reg. Transfer	<=	ax **<=** bx
Addition	+	ax **+** bx
Subtration	-	ax **-** bx
Shift Left	sll	ax **sll** 2
Shift Right	srl	ax **srl** 2

	Operator	Example
Bitwise AND	and	ax **and** bx
Bitwise OR	or	ax **or** bx
Bitwise NOT	not	**not** ax
Bitwise XOR	xor	ax **xor** bx
Vectors	ax(3 down to 0)	ax(3 down to 0)
Concatenate	&	ax(7 down to 4) **&**ax(3 down to 0)

Concept

Register Transfer VHDL

```
1) ax <= 12
2) bx <= 18
3) cx <= ax + bx
4) dx <= bx - ax
```

- What are the binary values after these operations? (Assuming 8bit registers)

- Answer:

```
ax: 0000 1100
bx: 0001 0010
cx: 0001 1110
dx: 0000 0110
```

Example

Register Transfer VHDL

```
1) ax <= 12
2) bx <= ax sll 2
3) cx <= ax sll 3
4) dx <= ax srl 1
```

- What are the binary values after these operations? (Assuming 8bit registers)

- Answer:

```
ax: 0000 1100
bx: 0011 0000
cx: 0110 0000
dx: 0000 0110
```

Example

Register Transfer VHDL

```
1) ax <= 12
2) bx <= 20
3) cx <= ax and bx
4) dx <= ax or bx
```

- What are the binary values after these operations? (Assuming 8bit registers)

- Answer:

```
ax: 0000 1100
bx: 0001 0100
cx: 0000 0100
dx: 0001 1100
```

Example

Register Transfer VHDL

```
1) ax <= 12
2) bx <= 20
3) cx <= not ax
4) dx <= ax xor bx
```

- What are the binary values after these operations? (Assuming 8bit registers)

- Answer:

```
ax: 0000 1100
bx: 0001 0100
cx: 1111 0011
dx: 0001 1000
```

Example

Register Transfer VHDL

```
1) ax <= 12
2) bx <= 20
3) cx <= ax(5 down to 2)
4) dx <= bx(7 down to 4)
5) bx <= ax(5 down to 2) & bx(7 down to 4)
```

- What are the binary values after these operations? (Assuming 8bit registers)
- Answer:

```
ax: 0000 1100
bx: 0001 0100
cx: 0000 0011
dx: 0000 0001
bx: 0011 0001
```

Exercise

Register Transfer VHDL

```
1) ax <= 8
2) bx <= 23
3) cx <= bx(7 down to 4) & a(3 down to 0)
4) dx <= ax(4 down to 1) sll 4
```

- What are the binary values after these operations? (Assuming 8bit registers)
- Answer:

```
ax: 0000 1000
bx: 0001 0111
cx: 0001 1000
dx: 0100 0000
```

Exercise

Register Transfer VHDL

```
1) ax <= 13
2) bx <= 27
3) cx <= bx srl 2
4) dx <= (ax and bx) xor (not cx)
```

- What are the binary values after these operations? (Assuming 8bit registers)
- Answer:

```
ax: 0000 1101
bx: 0001 1011
cx: 0000 0110
dx: 1111 0000
```

Exercise

Register Transfer VHDL

```
1) ax <= 2Eh
2) bx <= ax(7 down to 4) xor ax(3 down to 0)
3) cx <= (ax slr 2) or (ax sll 1)
4) dx <= bx and cx
```

- What are the binary values after these operations? (Assuming 8bit registers)
- Answer:

```
ax: 0010 1110
bx: 0000 1100
cx: (0000 1011) or (0101 1100): (0101 1111)
dx: 0000 1100
```

Exercise

P3
Transferring

Register Transfer Operations

	Operator	Example
Assignment	`<=`	`ax <= 12h`
Reg. Transfer	`<=`	`ax <= bx`
Addition	`+`	`ax + bx`
Subtration	`-`	`ax - bx`
Shift Left	`sll`	`ax sll 2`
Shift Right	`srl`	`ax srl 2`

	Operator	Example
Bitwise AND	`and`	`ax and bx`
Bitwise OR	`or`	`ax or bx`
Bitwise NOT	`not`	`ax not bx`
Bitwise XOR	`xor`	`ax xor bx`
Vectors	`ax(3 down to 0)`	`ax(3 down to 0)`
Concatenate	`&`	`ax(7 down to 4)` `&ax(3 down to 0)`

Review