COE 530
Quantum Computer And Architecture

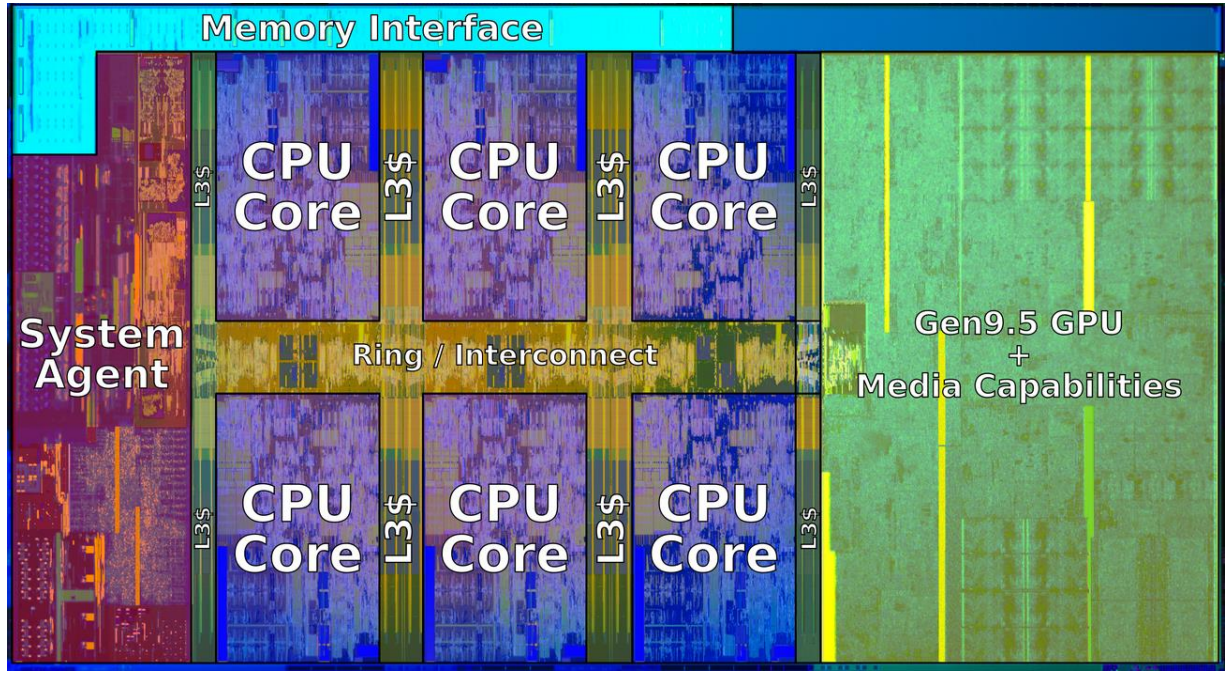
Lecture 4

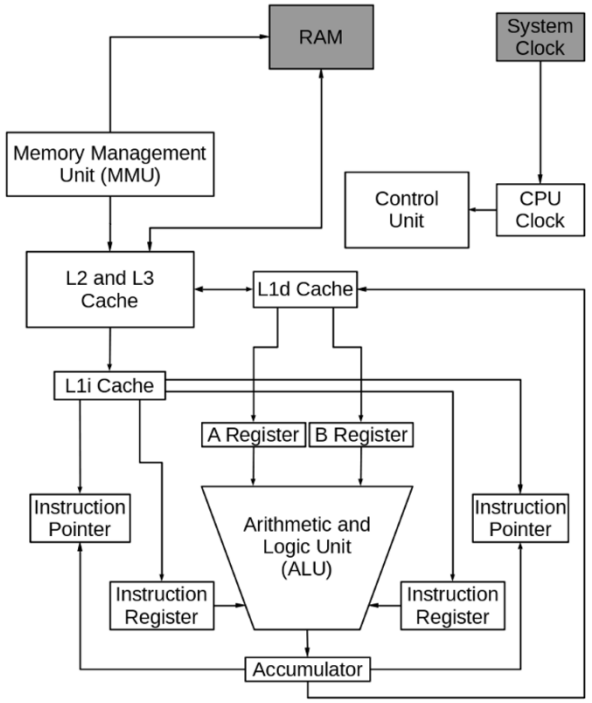
Classical Computer System V

References:

Quantum Operating Systems https://arxiv.org/pdf/2406.19120

State of the art CPU





https://www.redhat.com/sysadmin/cpu-components-functionality

This slide is from CMU course

Definitions

- Architecture: (also ISA: instruction set architecture) The parts of a processor design that one needs to understand for writing assembly/machine code.
  - Examples: instruction set specification, registers
- Microarchitecture: Implementation of the architecture
  - Examples: cache sizes and core frequency
- Code Forms:
  - Machine Code: The byte-level programs that a processor executes
  - Assembly Code: A text representation of machine code
- Example ISAs:
  - Intel: x86, IA32, Itanium, x86-64
  - ARM: Used in almost all mobile phones
  - RISC V: New open-source ISA

This slide is from CMU course

Levels of Abstraction

C programmer

```c
#include <stdio.h>

int main() {
    int a, b, sum;

    a = 5;
    b = 7;
    sum = a + b;

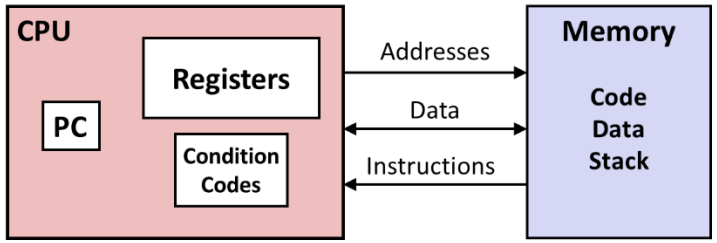
    printf("Sum = %d\n", sum);
    return 0;
}
```

```python
a = 5
b = 7
sum = a + b

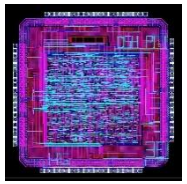
print("Sum =", sum)
```

Python programmer

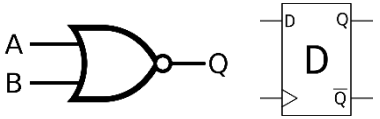
Assembly programmer



Computer Designer



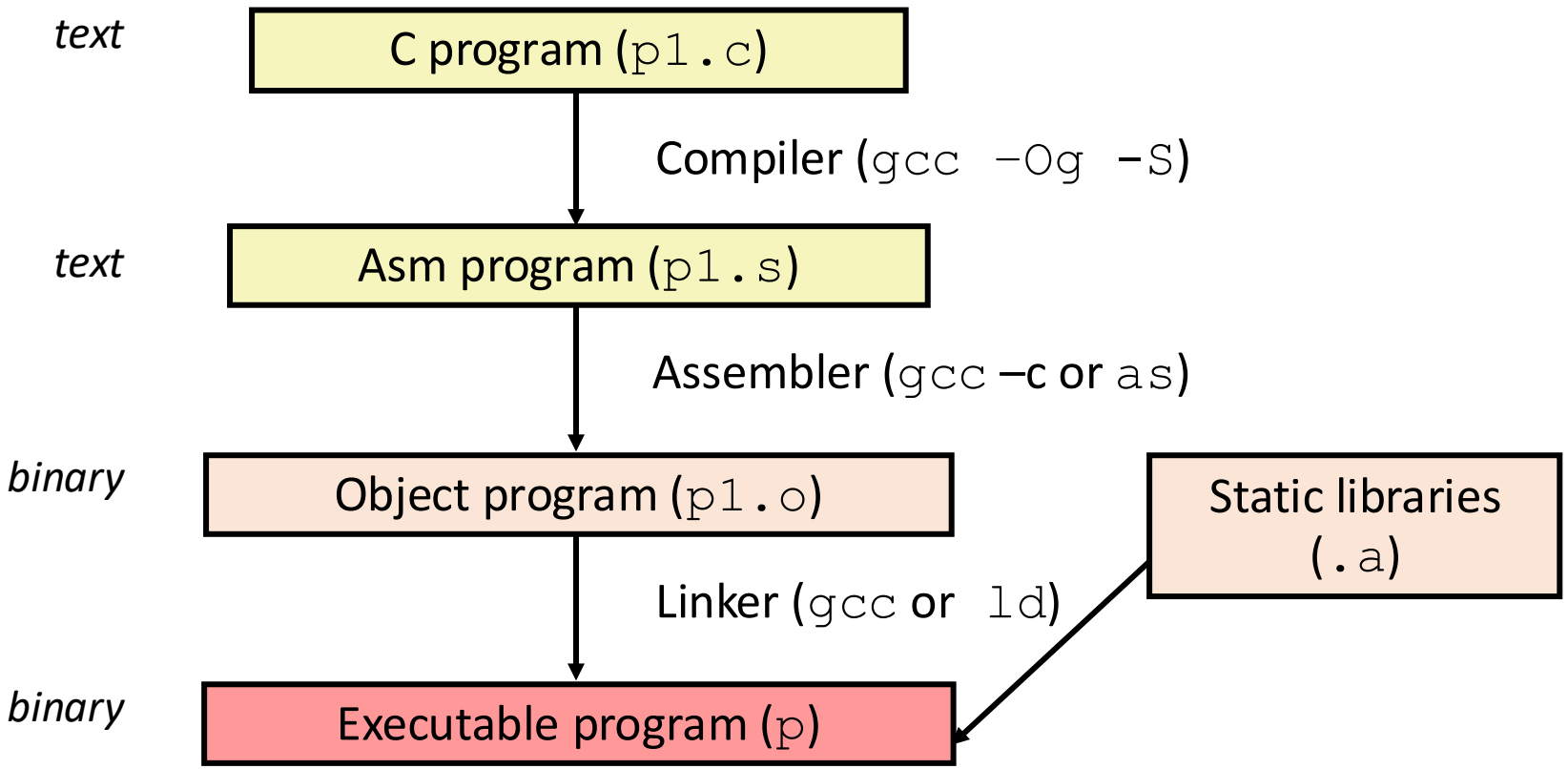
Gates, clocks, circuit layout, …



This slide is from CMU course

Turning C into Object Code

- Code in files p1.c
- Compile with command: gcc –Og p1.c
  - Put resulting binary in file p



This slide is from CMU course

Compiling Into Assembly

- C Code (sum.c)

```
#include <stdio.h>
int main() {
    int a, b, sum;
    a = 5; b = 7;
    sum = a + b;
    printf("Sum = %d\n", sum);
    return 0;
}
```

Generated x86-64 Assembly

```
..
    movl    $7, -8(%rbp)
    movl    -4(%rbp), %eax
    addl    -8(%rbp), %eax
    movl    %eax, -12(%rbp)
    movl    -12(%rbp), %esi
..
```

Obtain (on shark machine) with command

```
gcc –Og –S sum.c
```

Produces file `sum.s`

Warning: Will get very different results on non-Shark machines (Andrew Linux, Mac OS-X, …) due to different versions of gcc and different compiler settings.

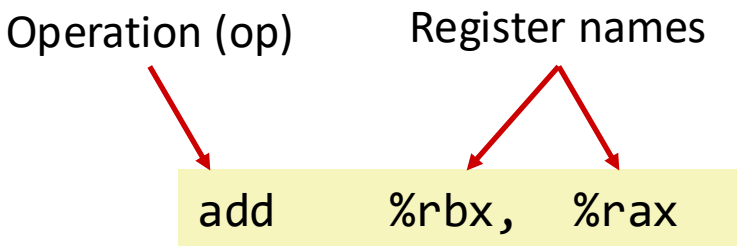
What it really looks like

```
        .globl main
main:
    pushq   %rbp
    movq    %rsp, %rbp
    movl    $5, -4(%rbp)
    movl    $7, -8(%rbp)
    movl    -4(%rbp), %eax
    addl    -8(%rbp), %eax
    movl    %eax, -12(%rbp)
    movl    -12(%rbp), %esi
    leaq    .LC0(%rip), %rdi
    call    printf
    movl    $0, %eax
    popq    %rbp
    ret
```

This slide is from CMU course

Beginning Grammar

- Say something in assembly:



is

```
rax += rbx
```

Nouns

- Registers
  - Special parts of the CPU to hold small amounts of data, like local variables
  - Names are very specific:

    %r?x (a,b,c,d)

    %r?i (d,s)

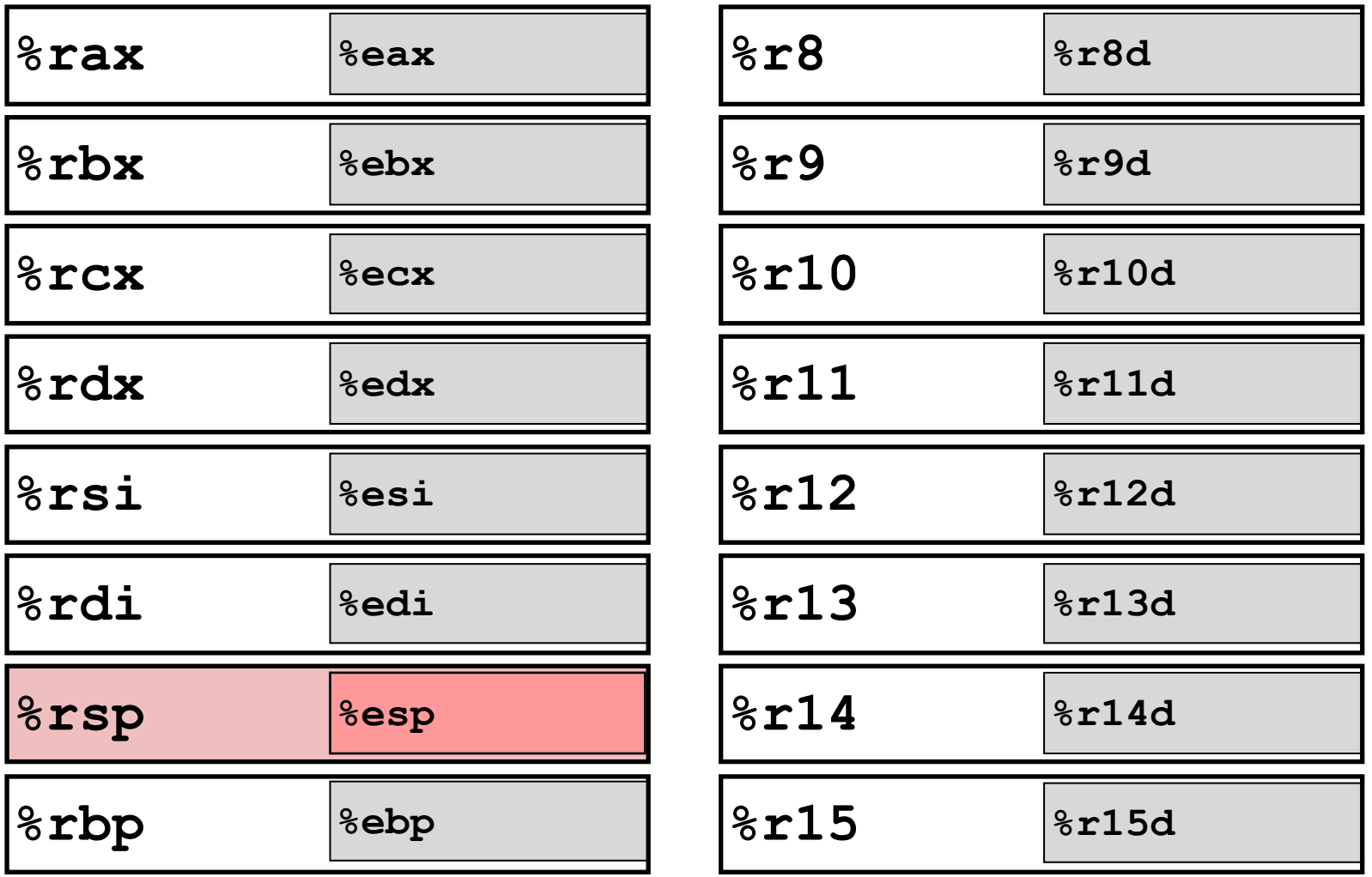
    %r?p (b,i,s) // Special meanings

    %r? (8-15)

- Memory
  - Everything else; accessed by *address*.

This slide is from CMU course

x86-64 Integer Registers



This slide is from CMU course

Verbs (i.e., operations)

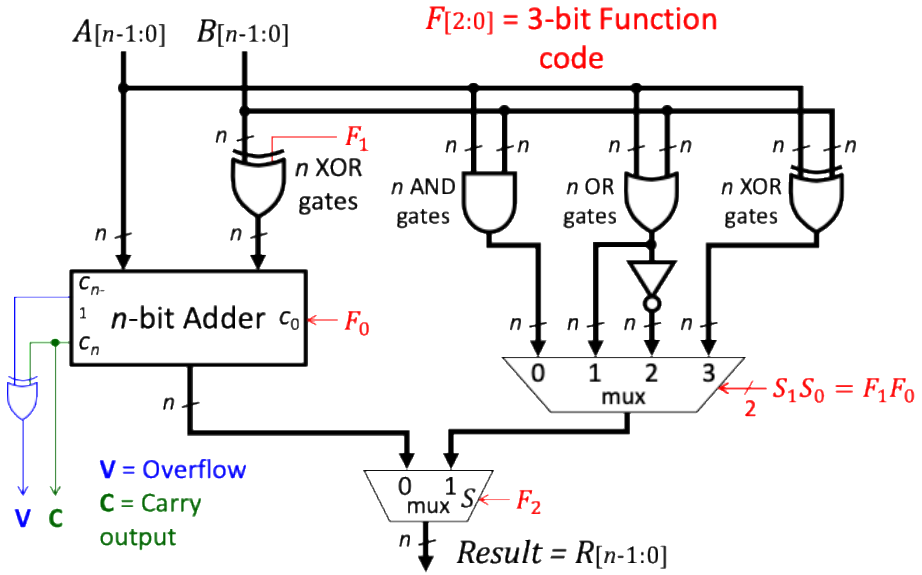
- Transfer data between memory and register
  - Load data from memory into register
  - Store register data into memory
- Perform arithmetic function on register or memory data
- Transfer control
  - Unconditional jumps to/from procedures
  - Conditional branches
  - Indirect branches

This slide is from CMU course

Object Code

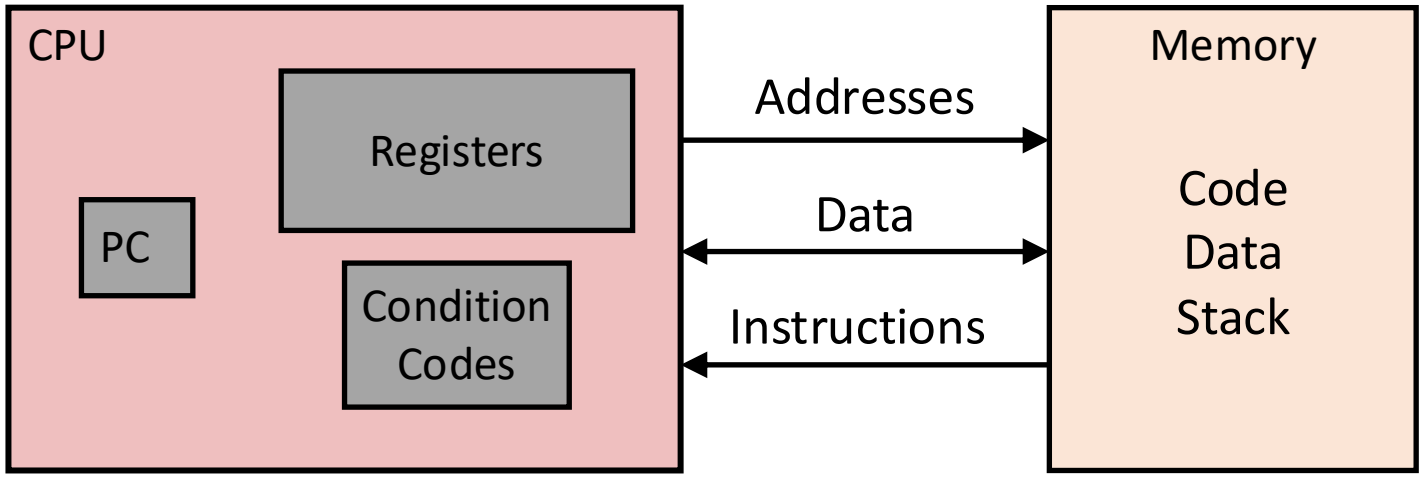
```
0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   c7 45 fc 05 00 00 00    movl   $0x5,-0x4(%rbp)
   b:   c7 45 f8 07 00 00 00    movl   $0x7,-0x8(%rbp)
  12:   8b 45 fc                mov    -0x4(%rbp),%eax
  15:   03 45 f8                add    -0x8(%rbp),%eax
```

add has op code (03)



This slide is from CMU course

Assembly/Machine Code View



- Programmer-Visible State
  - PC: Program counter
    - Address of next instruction
    - Called “RIP” (x86-64)
  - Register file
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic or logical operation
    - Used for conditional branching

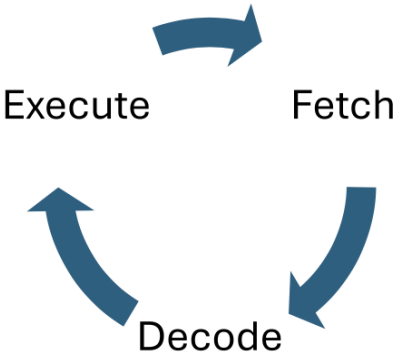
- Memory
  - Byte addressable array
  - Code and user data
  - Stack to support procedures

This slide is from CMU course

Fetch-Decode-Execute Cycle in CPU

- The basic process a CPU uses to run every instruction in a program.
- **Fetch**
  - The CPU **retrieves the next instruction** from main memory (RAM)
  - Instruction address comes from the **Program Counter (PC)**
  - The instruction is stored in the **Instruction Register (IR)**
  - PC is updated to point to the next instruction
- **Decode**
  - The **Control Unit** interprets the instruction
  - Determines:
    - What operation to perform
    - Which data/registers are needed
    - Prepares signals for execution
- **Execute**
  - The CPU **performs the operation**:
  - Arithmetic / logic (via ALU)
  - Memory access
  - Data movement
  - Results are stored in registers or memory

Fig. 4.16
Fetch–Decode–Execute cycle



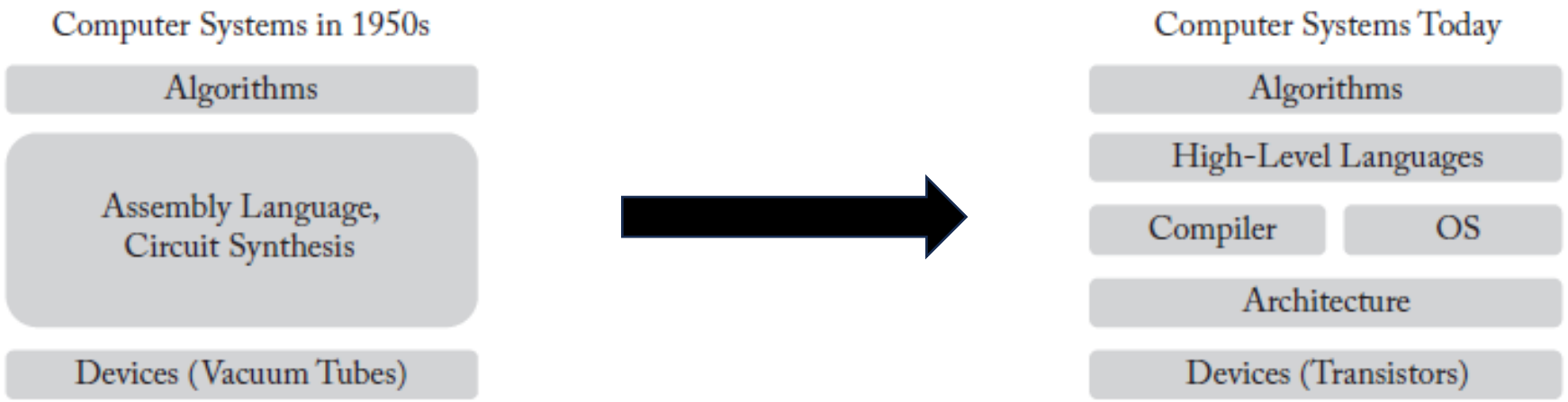
The Physics of Computing

Exercise

- Play with the Little Man Computer

https://wellingborough.github.io/LMC/LMC0.3.html

Today's computers are much much more complex than LMC

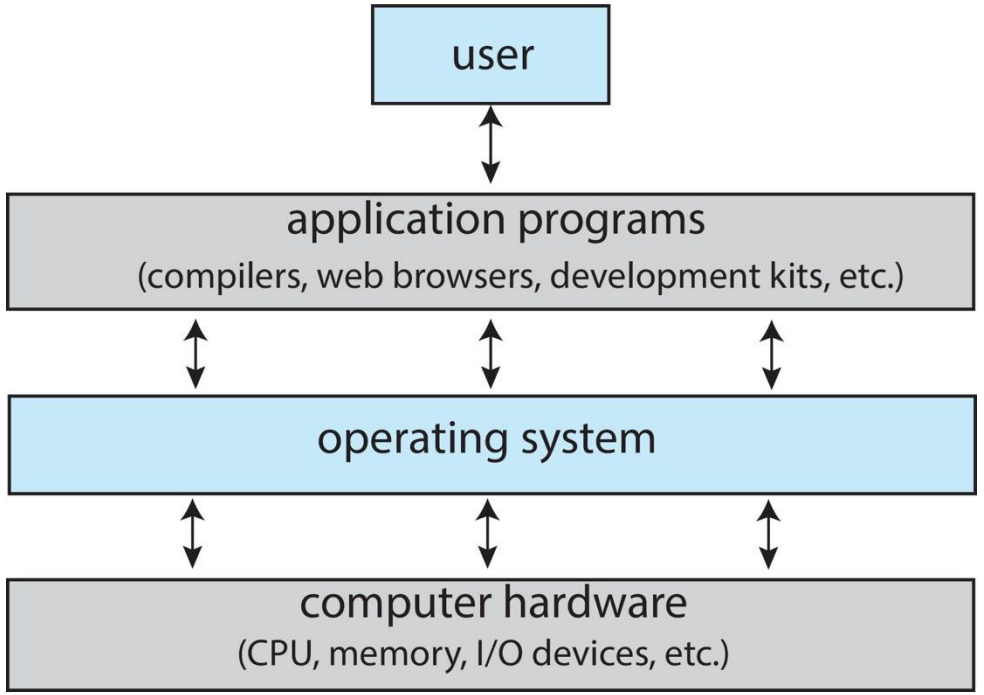


Questions for reflection

- What is quantum architcutre?
- What would be the qunatum instruction sets?
- What is ”quantum” object code?
- At which levels are errors handled in classical computer architecture?
- Can we run mulitple quantum circuits on the same QPU simultanously?
- Where are qubits stored to enable long-duration quantum computations?
- Where does the compilation of a quantum circuit take place?

What is an Operating System?

- A program that acts as an intermediary between a user of a computer and the computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner
- OS Functionalities
  - Processes management (threads and concurrency)
  - Memory management (virtualization)
  - CPU scheduling
  - File system
  - Security



Quantum OS

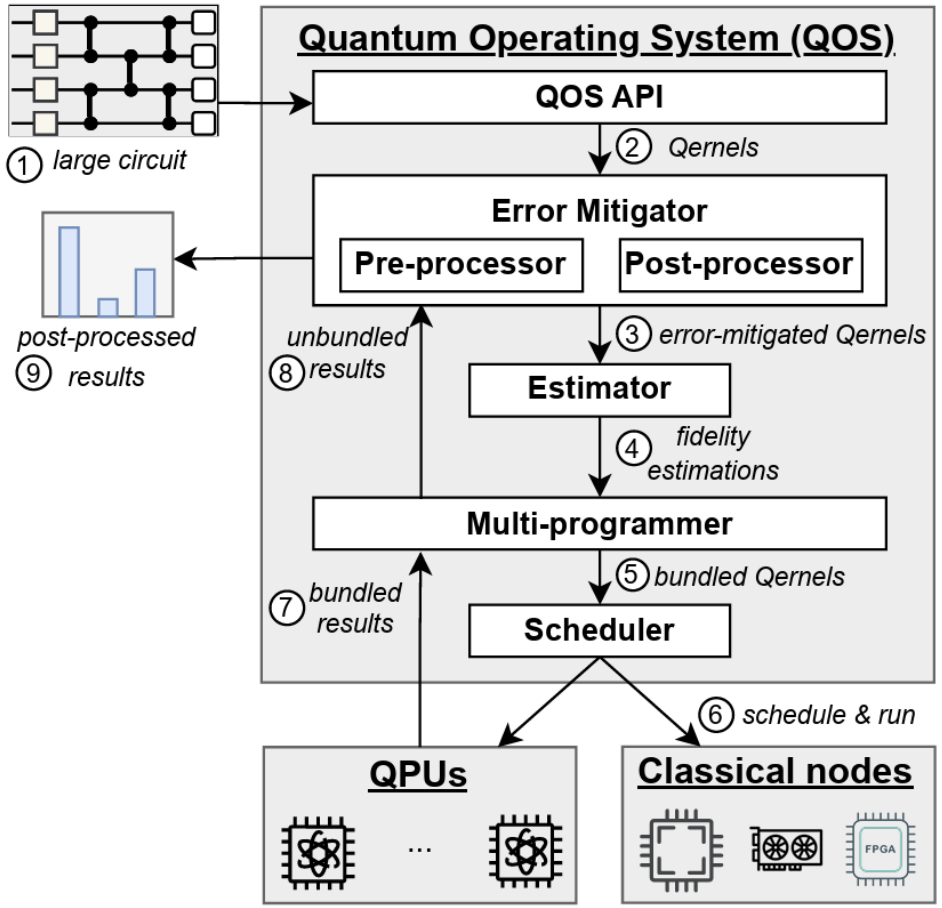


Figure 5: QOS overview (§ 4): *QOS consists of four main components: the error mitigator, estimator, multi-programmer, and scheduler.*

https://arxiv.org/pdf/2406.19120

QOS
Quantum Operating System

Emmanouil (Manos) Giortamis, Francisco Romão,
Nathaniel Tornow, Pramod Bhatotia
Technical University of Munich

USENIX OSDI 2025, Boston, USA

https://www.youtube.com/watch?v=AJB5aGrYTf0