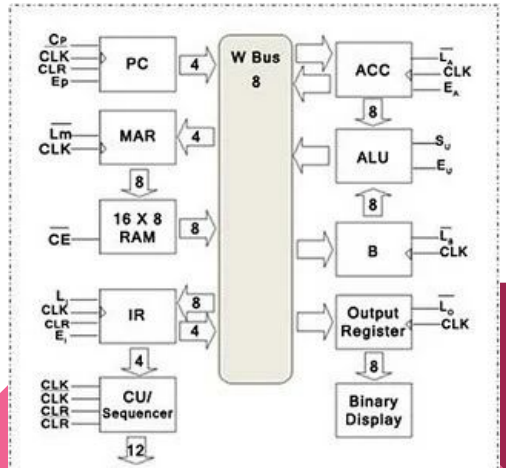
CS 24000 Week 13

How CPUs work from a low level,
How functions work from a high level

CPU Instructions, registers and pointers

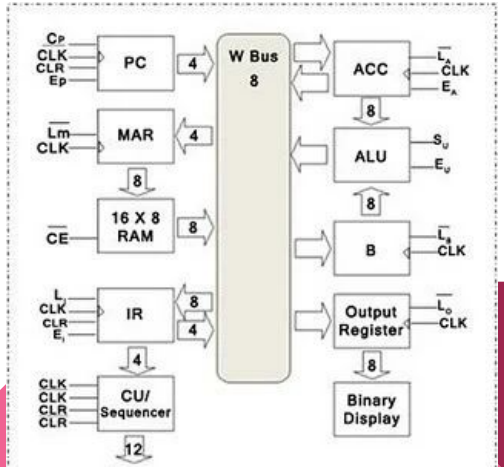
- We'll start with an 8 bit CPU with only a handful of instructions
  - This can be generalized to more modern CPUs



An example 8-bit CPU

CPU Instructions, registers and pointers

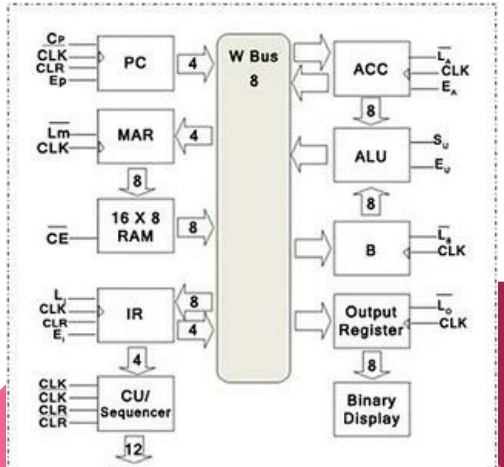
- We'll start with an 8 bit CPU with only a handful of instructions
  - This can be generalized to more modern CPUs
- CPUs have instructions, which are basic things like add, multiply, &, |, etc.
  - Example: Add r1, r2 adds r2 to r1, saving the value to r1
  - r1 and r2 are **registers**, which are like RAM, but there are only a few of them (30-40 on most processors today)
  - Most instructions use only two registers



An example 8-bit CPU

CPU Instructions, registers and pointers (cont.)

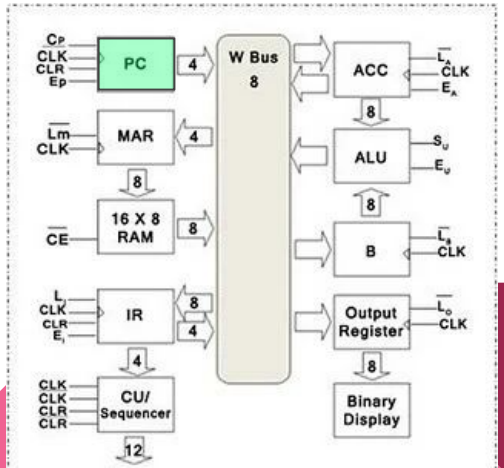
- More advanced instructions exist to reference and dereference memory
  - Example: Load r1, 0xf1 loads the value of address 0xf1 into register r1
  - Store r1, 0xf2 stores r1's value into 0xf2



An example 8-bit CPU

CPU Instructions, registers and pointers (cont.)

- More advanced instructions exist to reference and dereference memory
  - Example: Load r1, 0xf1 loads the value of address 0xf1 into register r1
  - Store r1, 0xf2 stores r1's value into 0xf2
- Some registers handle important stuff
  - PC, or the program counter, points to the next instruction in memory
  - Some registers are designated for return values, conditionals, etc.

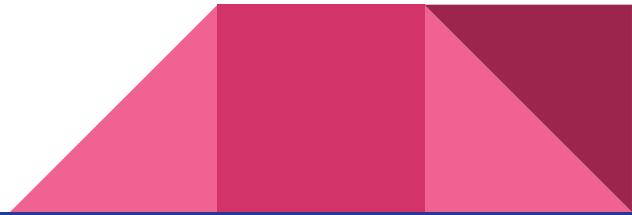


An example 8-bit CPU

What does this have to do with HW12??

- One special register, PC, is responsible for keeping track of the current state of the program
  - When you call a function, that function's memory address is stored into PC, and the remaining registers are filled with the function's input arguments
- PC can also jump to a function pointer, which behaves exactly the same
  - If you have a function *int fn(int x)*, and a function pointer *int (*fn_ptr)(int) = &fn*, calling either of these will update the program counter to the same location

Side note: The stack exists in memory to keep track of variables that don't fit in the registers (and the heap stores dynamic memory)



Functions in the Heap/Stack

- It is technically possible to store functions in main memory
  - Why wouldn't you though?
- It's bad practice, but I'll talk about it anyway
- It requires an extra flag when compiling
  - gcc -zexecstack …
- Typically all we care about is the function pointer itself
  - Storing the actual function in main memory is useful for self modifying code, but that's a topic for another day
  - See "execstack_sample.c" on my personal website for a simple example