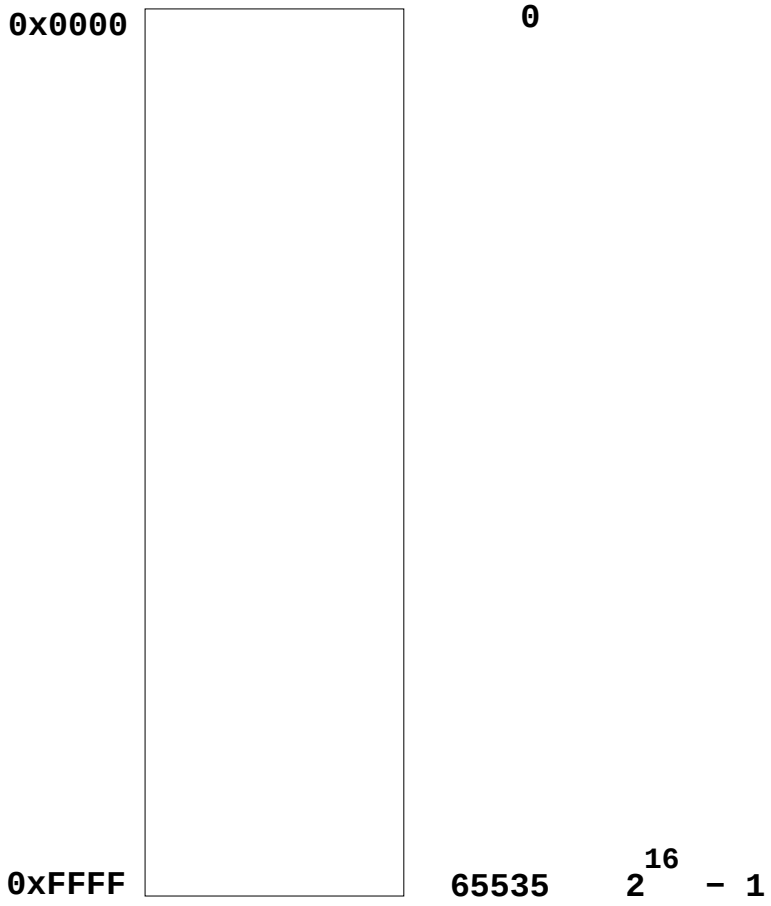
68HC12 Address Space

- 68HC12 has 16 address lines
- 68HC12 can address 2^{16} distinct locations
- For 68HC12, each location holds one byte (eight bits)
- 68HC12 can address 2^{16} bytes
- $2^{16} = 65536$
- $2^{16} = 2^6 \times 2^{10} = 64 \times 1024 = 64$ KB
- $(1\text{K} = 2^{10} = 1024)$
- 68HC12 can address 64 KB

68HC12 Address Space

- Lowest address: $0000000000000000_2 = 0000_{16} = 0_{10}$
- Highest address: $1111111111111111_2 = \mathrm{FFFF}_{16} = 65535_{10}$



MEMORY TYPES

RAM: Random Access Memory (can read and write)

ROM: Read Only Memory (programmed at factory)

PROM: Programmable Read Only Memory
(Program once at site)

EPROM: Erasable Programmable Read Only Memory
(Program at site, can erase using UV light and reprogram)

EEPROM: Electrically Erasable Programmable Read Only Memory
(Program and erase using voltage rather than UV light)

68HC12 has: 1 K RAM

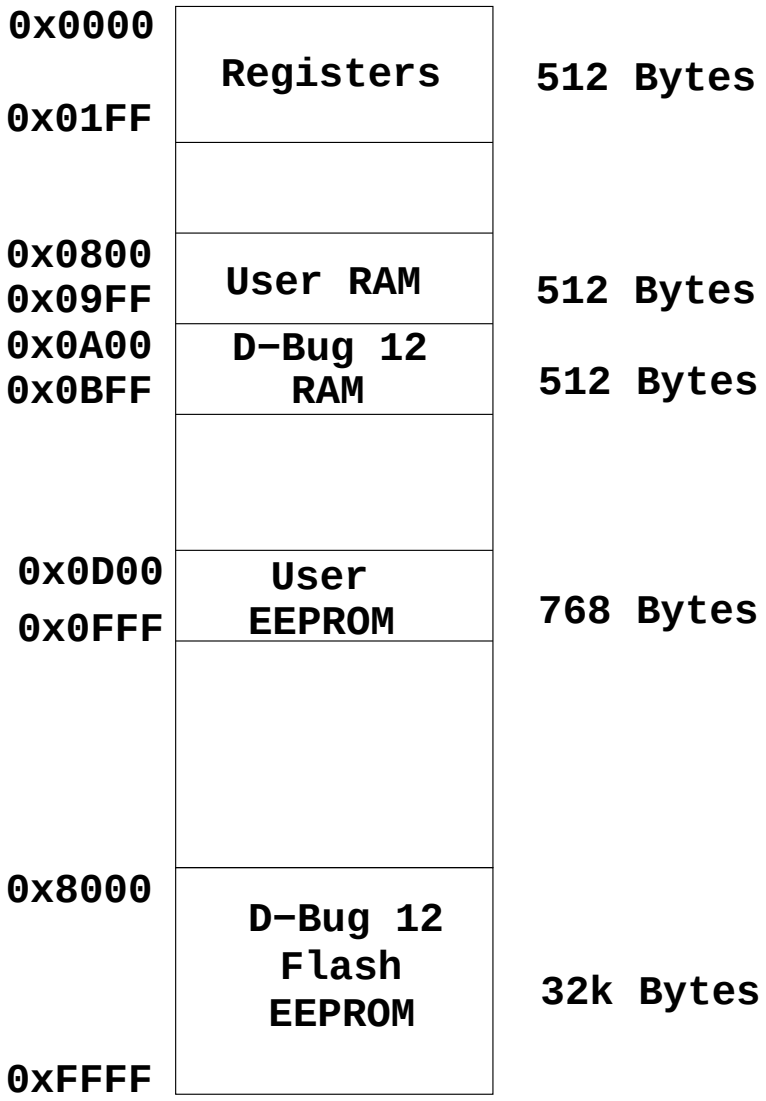
768 bytes EEPROM

Can erase and reprogram any byte using normal 5V pow

32 KB Flash EEPROM

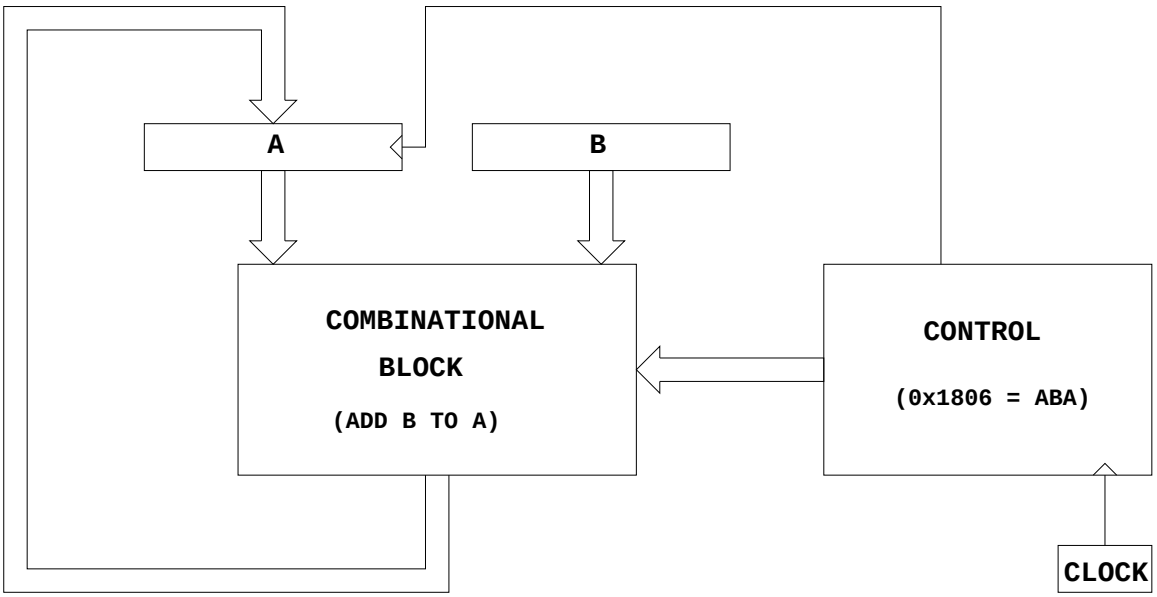
Can erase using external 12V power supply

68HC12 Address Space



68HC12 ALU

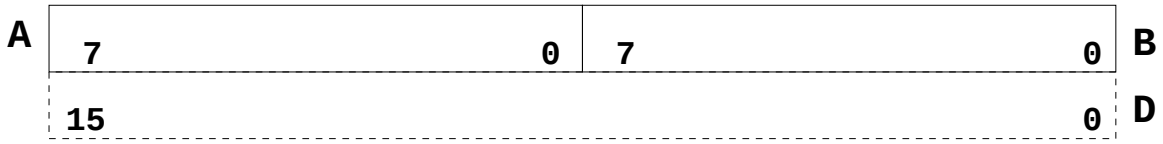
- Arithmetic Logic Unit (ALU) is where instructions are executed.
- Examples of instructions are arithmetic (add, subtract), logical (bitwise AND, bitwise OR), and comparison.
- 68HC12 has two 8-bit registers for executing instructions. These registers are called **A** and **B**.
- For example, the HC12 can add the 8-bit number stored in **B** to the eight-bit number stored in **A** using the instruction **ABA** (add B to A):



When the control unit sees the sixteen-bit number `0x1806`, it tells the ALU to add **B** to **A**, and store the result into **A**.

68HC12 Programming Model

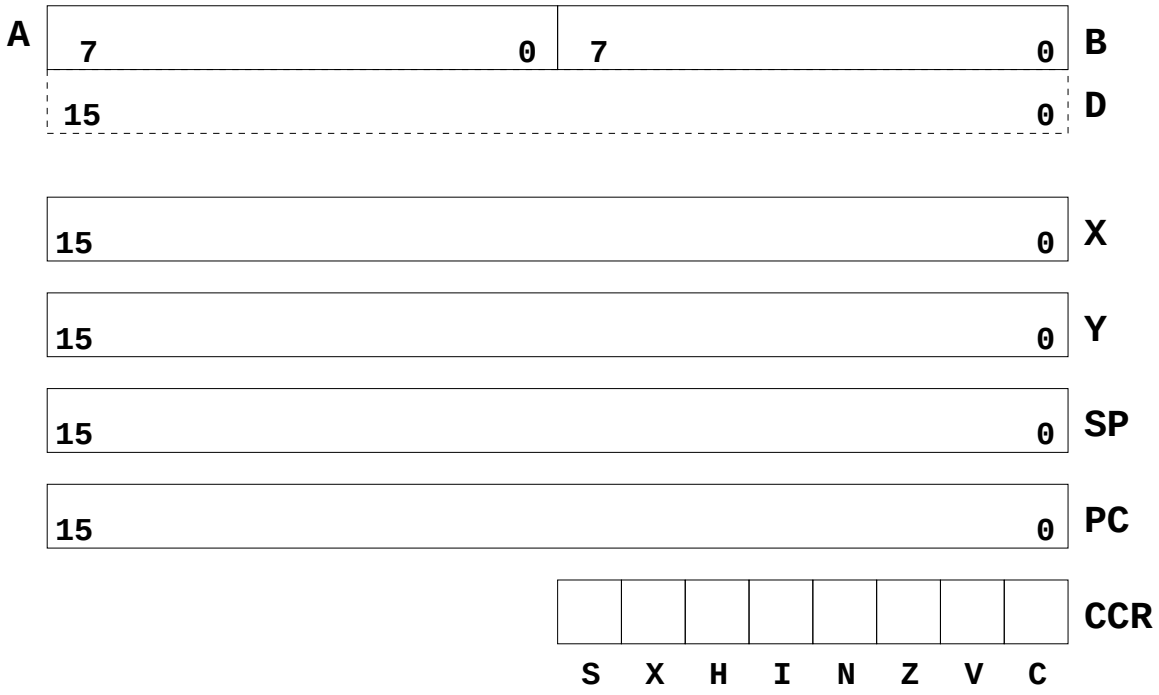
- A Programming Model details the registers in the ALU and control unit which a programmer needs to know about to program a microprocessor.
- Registers **A** and **B** are part of the programming model. Some instructions treat **A** and **B** as a sixteen-bit register called D for such things as adding two sixteen-bit numbers. Note that D is the same as **A** and **B**.



- The HC12 can work with 8-bit numbers (bytes) and 16-bit numbers (words).
- The size of word the HC12 uses depends on the instruction. For example, the instruction LDAA (Load Accumulator A) puts a byte into **A**, and LDD (Load Double Accumulator) puts a word into **D**.

68HC12 Programming Model

- The 68HC12 has a sixteen-bit register which tells the control unit which instruction to execute. This is called the Program Counter (**PC**). The number in **PC** is the address of the next instruction the HC12 will execute.
- The 69HC12 has an eight-bit register which tells the HC12 about the state of the ALU. This register is called the Condition Code Register (CCR). For example, one bit (C) tells the HC12 whether the last instruction executed generated a carry. Another bit (Z) tells the HC12 whether the result last instruction was zero. The N bit tells whether the last instruction executed generated a negative result.
- There are three other 16-bit registers – X, Y, SP – which we will discuss later.



Some HC12 Instructions Needed for Lab 1

`LDAA address`	Put the byte contained in memory at `address` into **A**
`STAA address`	Put the byte contained in **A** into memory at `address`
`CLRA`	Clear **A** (0 -> **A**)
`INCA`	Add 1 to **A** ((**A**) + 1 -> **A**)
`ABA`	Add **B** to **A**, store the result in **A**
`ASRA`	Shift **A** right by one bit (keep the MSB the same) This divides a signed byte by 2
`LSRA`	Shift **A** right by one bit (put 0 into MSB) This divides an unsigned byte by 2
`NEGA`	Negate **A** (-(**A**) -> **A**)
`TAB`	Transfer **A** to **B** ((**A**) -> **B**)
`SWI`	Software Interrupt (Used to end all our HC12 programs)

A Simple HC12 Program

- All programs and data must be placed in memory between address `0x0800` and `0x09FF`. For our short programs we will put the first instruction at `0x0800`, and the first data byte at `0x0900`
- Consider the following program:

```
ldaa $0913   ; Put contents of memory at 0x0913
inca         ; Add one to A
staa $0914   ; Store the result into memory at
swi          ; End program
```

- If the first instruction is at address `0x0800`, the following bytes in memory will tell the HC12 to execute the above program:

Address	Value
0x0800	B6
0x0801	09
0x0802	13
0x0803	42
0x0804	7A
0x0805	09
0x0806	14
0x0807	3F

- If the contents of address `0x0913` were `0xA2`, the program would put an `0xA3` into address `0x0914`.

A Simple Assembly Language Program.

- It is difficult for humans to remember the numbers (op codes) for computer instructions. It is also hard for us to keep track of the addresses of numerous data values. Instead we use words called mnemonics to represent instructions, and labels to represent addresses, and let a computer programmer called an assembler to convert our program to binary numbers (machine code).
- Here is an assembly language program to implement the previous program:

```
prog     equ      $0800  ; Start program at 0x0
data     equ      $0913  ; Data value at 0x0913
result   equ      $0914  ; Result at 0x0914
         org      prog
CODE:    section  .text
         ldaa     data
         inca
         staa     result
         swi
```

- We would put this code into a file and give it a name, such as `test.s`
- Note that `equ` ,`org`, and `section` are not instructions for the HC12 but are directives to the assembler which make it possible for us to write assembly language programs. The are called assembler directives or psuedo-ops. For example the psuedo-op `org` tells the assembler that the starting address (origin) of our program should be `0x0800`.

Assembling an Assembly Language Program

- A computer program called an assembler can convert an assembly language program into machine code.
- The assembler we use in class is from a company called Cosmic.
- To assemble the above program using the Cosmic assembler, we must first create a file called `test.lkf` which tells the assembler where to put things in memory. Our `test.lkf` file would look like this:

```
# Link file for test program
+seg .text -b 0x0800 -n .text  # program start
+seg .data -b 0x0800 -n .data  # data start add
test.o                         # application pr
```

- To assemble the program, use the following commands:

```
ca6812 -a -l -xx -pl test.s
clnk -o test.h12 -m test.map test.lkf
chex -o test.s19 test.h12
```

- This will produce a file called `test.s19` which we can load into the 68HC12.