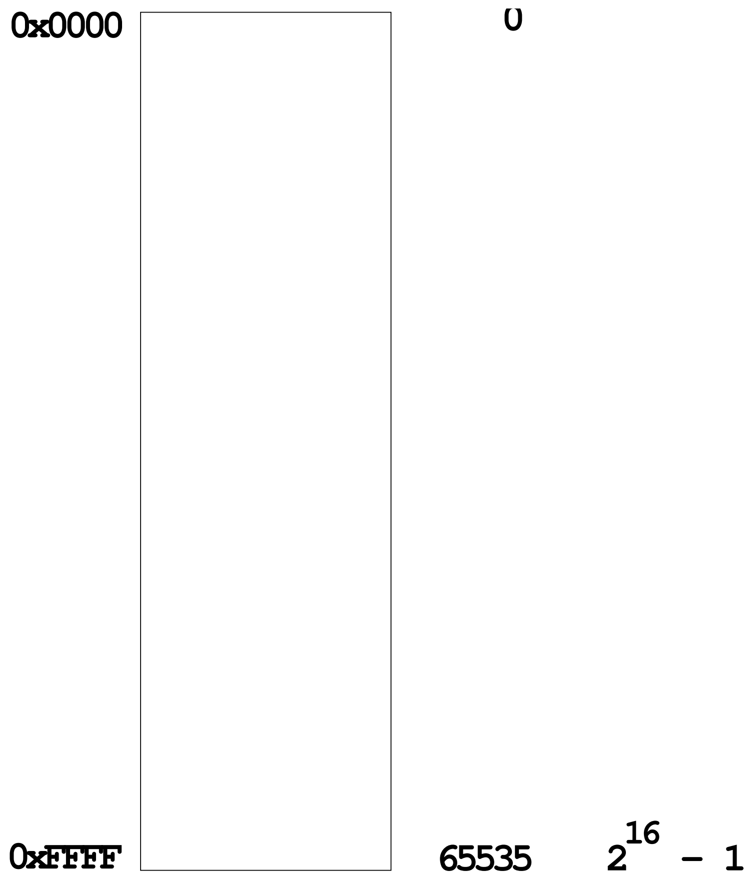
MC9S12 Address Space

- MC9S12 has 16 address lines
- MC9S12 can address 2^{16} distinct locations
- For MC9S12, each location holds one byte (eight bits)
- MC9S12 can address 2^{16} bytes
- $2^{16} = 65536$
- $2^{16} = 2^6 \times 2^{10} = 64 \times 1024 = 64$ KB
- $(1\text{K} = 2^{10} = 1024)$
- MC9S12 can address 64 KB

MC9S12 Address Space

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



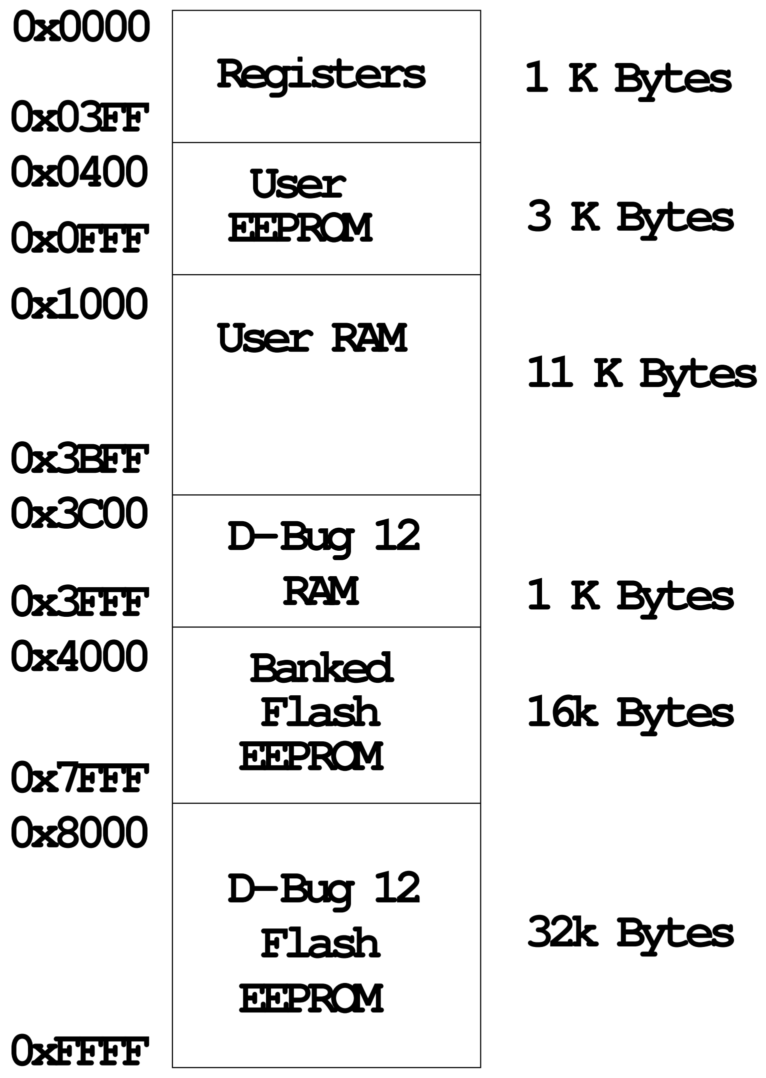
MEMORY TYPES

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)

MC9S12 has: 12 KB RAM

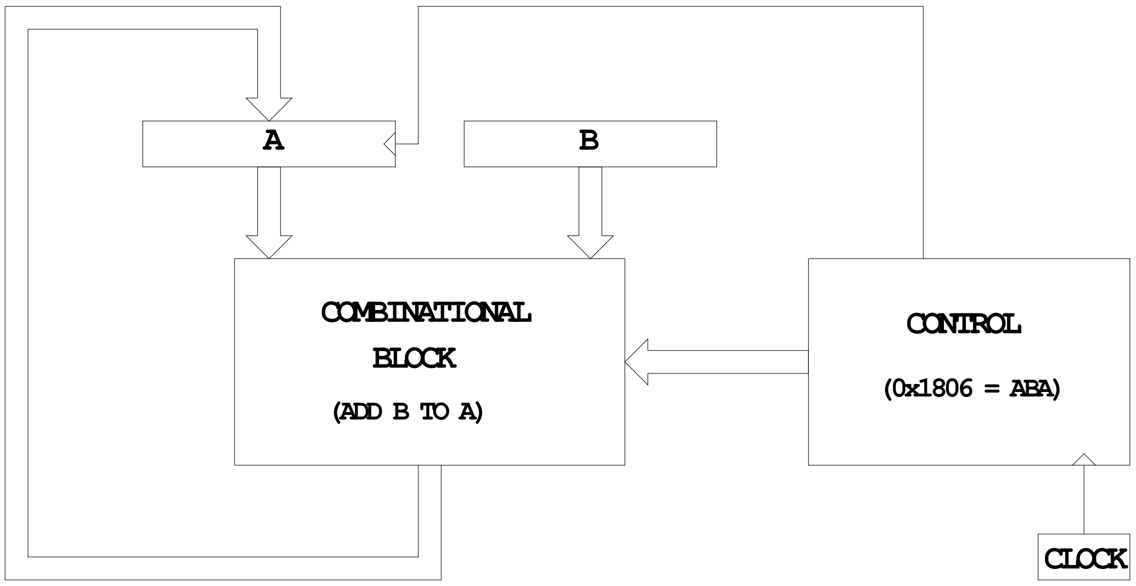
3 KB EEPROM
256 KB Flash EEPROM (Can access 48 KB at a time)

MC9S12 Address Space



MC9S12 ALU

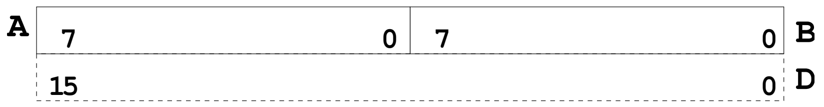
- Arithmetic Logic Unit (ALU) is where instructions are executed.
- Examples of instructions are arithmetic (add, subtract), logical (bitwise AND, bitwise OR), and comparison.
- MC9S12 has two 8-bit registers for executing instructions. These registers are called **A** and **B**.
- For example, the MC9S12 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**.

MC9S12 Programming Model

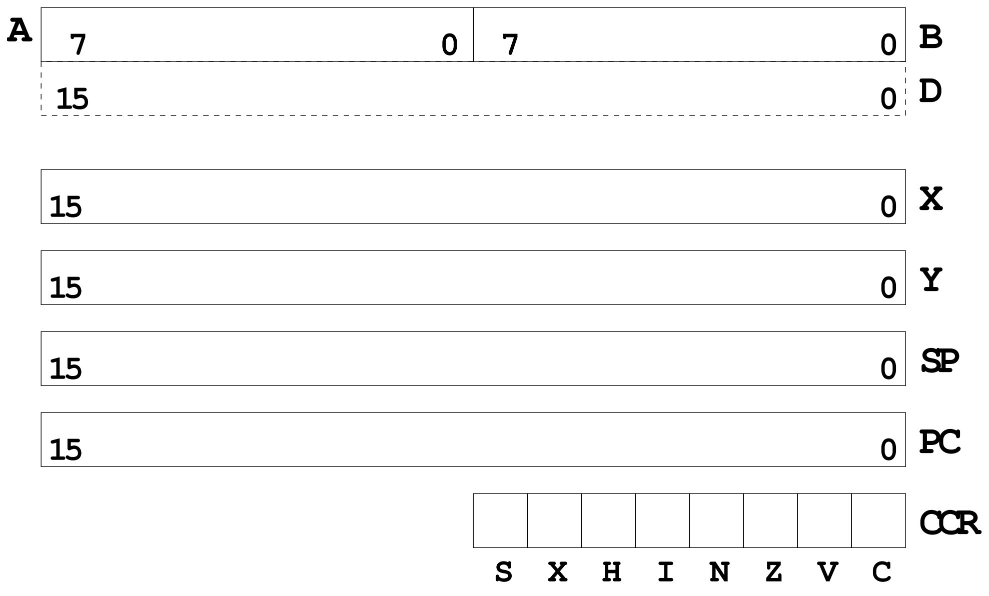
- A Programming Model details the registers in the ALU and control unit which a programmer needs to know about to program a microprocessor.
- Registers **A** and **B** are part of the programming model. Some instructions treat **A** and **B** as a sixteen-bit register called D for such things as adding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



- The MC9S12 can work with 8-bit numbers (bytes) and 16-bit numbers (words).
- The size of word the MC9S12 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**.

MC9S12 Programming Model

- The MC9S12 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 MC9S12 will execute.
- The MC9S12 has an eight-bit register which tells the MC9S12 about the state of the ALU. This register is called the Condition Code Register (CCR). For example, one bit (C) tells the MC9S12 whether the last instruction executed generated a carry. Another bit (Z) tells the MC9S12 whether the result of the 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 MC9S12 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 MC9S12 program

A Simple MC9S12 Program

- All programs and data must be placed in memory between address `0x1000` and `0x3BFF`. For our short programs we will put the first instruction at `0x1000`, and the first data byte at `0x2000`
- Consider the following program:

```
ldaa $2000 ; Put contents of memory at 0x2000 into A
inca       ; Add one to A
staa $2001 ; Store the result into memory at 0x2001
swi        ; End program
```

- If the first instruction is at address `0x1000`, the following bytes in memory will tell the MC9S12 to execute the above program:

Address	Value	Instruction
0x1000	B6	ldaa $2000
0x1001	20	
0x1002	00	
0x1003	42	inca
0x1004	7A	staa $2001
0x1005	20	
0x1006	01	
0x1007	3F	swi

- If the contents of address `0x2000` were `0xA2`, the program would put an `0xA3` into address `0x2001`.

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     equu     $1000  ; Start program at 0x1000
```

- We would put this code into a file and give it a name, such as `test.s`
- Note that `equ`, `org`, `dc.b` and `ds.b` are not instructions for the MC9S12 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 `0x1000`.

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 a freee compiler from Motorola.
- The easiest way to assemble it is to use the freeware IDE **AsmIDE**, as discussed in Lab 1 and in Huang.
- The assembler will produce a file called `test.lst`, which shows the machine code generated.

```
as12, an absolute assembler for Motorola MCU's, version 1.

1000                        prog     equ      $1000  ; Start p
2000                        data     equ      $2000  ; Data va

1000                                 org      prog

1000 b6 20 00                        ldaa     input
1003 42                              inca
1004 7a 20 01                        staa     result
1007 3f                              swi

2000                                 org      data
2000 a2                     input:   dc.b     $A2
2001                        result:  ds.b     1

Executed: Thu Jan 19 21:19:06 2006
Total cycles: 23, Total bytes: 9
Total errors: 0, Total warnings: 0
```

- This will produce a file called `test.s19` which we can load into the MC9S12.

```
S012000046696C653A20746573742E61736D0ADA
S10B1000B62000427A20013FF2
S1042000A239
S9030000FC
```

  - The first line of the S19 file starts with a S0: the S0 indicates that it is the first line.
  - The last line of the S19 file starts with a S9: the S9 indicates that it is the last line.
  - The other lines begin with a S1: the S1 indcates these lines are data to be loaded into the MC9S12 memory.
  - On the second line, the S1 if followed by a 0B. This tells the loader that there this line has 11 (0x0B) bytes of data follow.
  - The count 0B is followed by 1000. This tells the loader that the data should be put into memory starting with address 0x1000.
  - The next 16 hex numbers B62000427A20013F are the 8 bytes to be loaded into memory. You should be able to find these bytes in the `test.lst` file.
  - The last two hex numbers, 0xF2, is a one byte checksum, which the loader can use to make sure the data was loaded correctly.