Lab on IIC Bus

- Next week's lab
  1. Communicate with Dallas Semiconductor DS 1307 Real Time Clock
     (a) Set time and date in clock
     (b) Read time and date from clock and display
  2. Display time and date on LCD display
- Hardest program this semester
- Need to use functions
- How to write to LCD display done for you
  - Program `eg07_08.c` on textbook CD-ROM

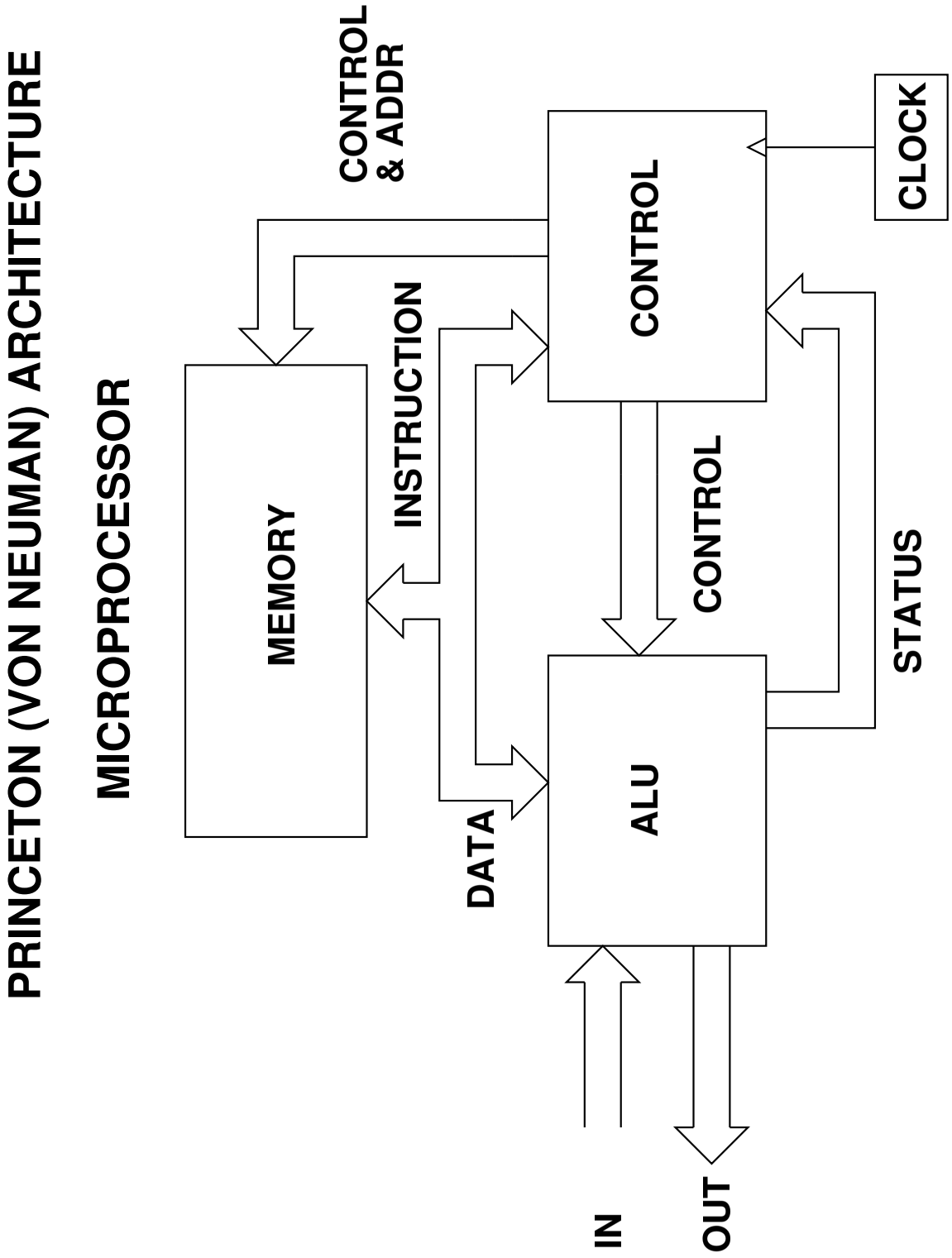
```
char msg[] = "hello, world!";
openlcd();
puts2lcd(msg);
```

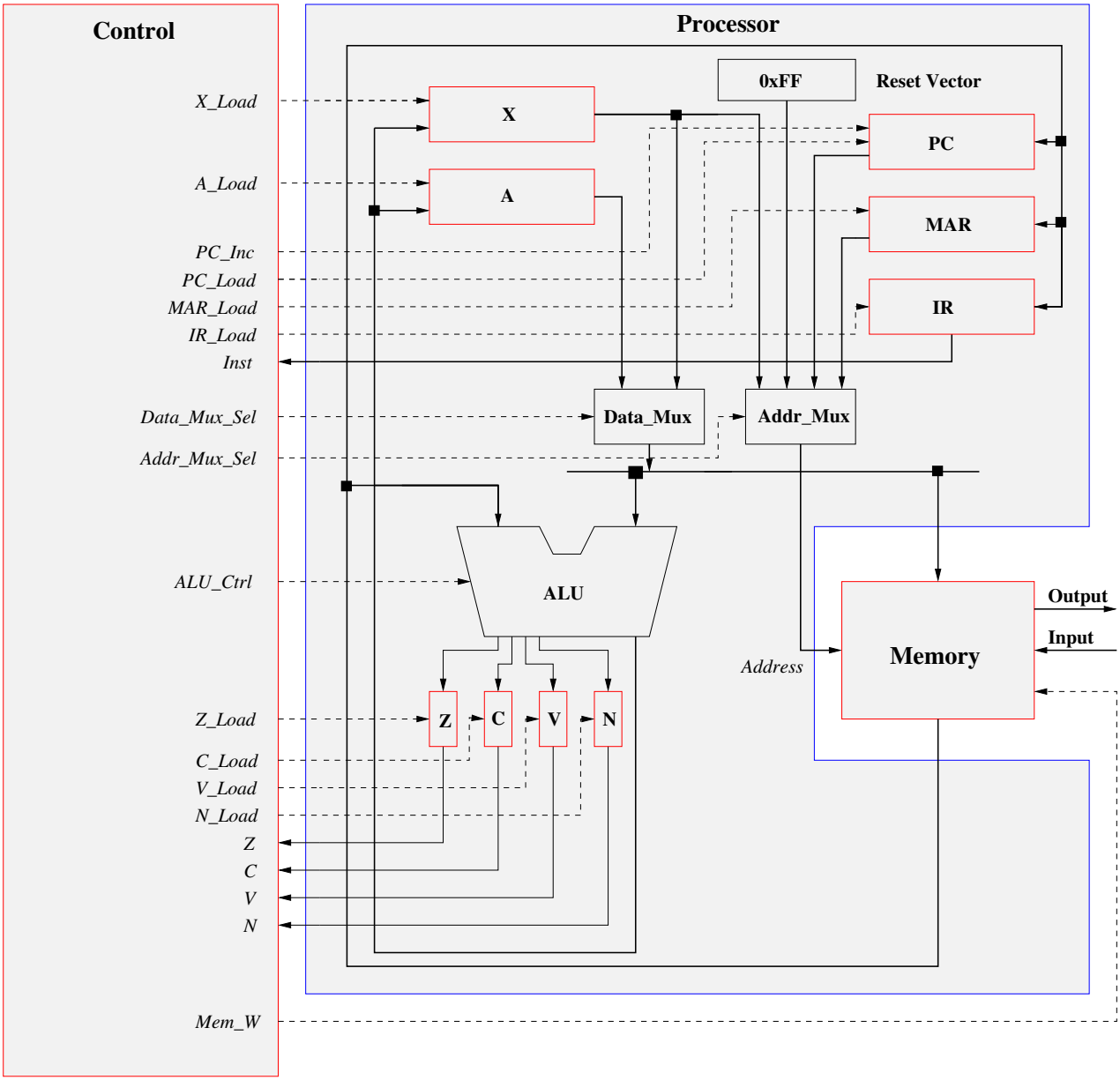
- Need to write functions to write to and read from RTC over the IIC bus
- Notes from March 23 have functions to initialize IIC bus, start a transfer by writing address and R/$\overline{\text{W}}$ bit, transmit a byte of data, and stop the transfer (release IIC bus).
- Need functions to switch to receive mode (`iic_swrcv()`) and receive data over IIC bus (`iic_receive`).
- Need to put functions together to write to the RTC, read from the RTC, and display the time/date on the LCD display
- To write data to LCD display, data has to be in the form of an ASCII string
- Data from RTC is in form of BCD data
- For example, year is `0x09`

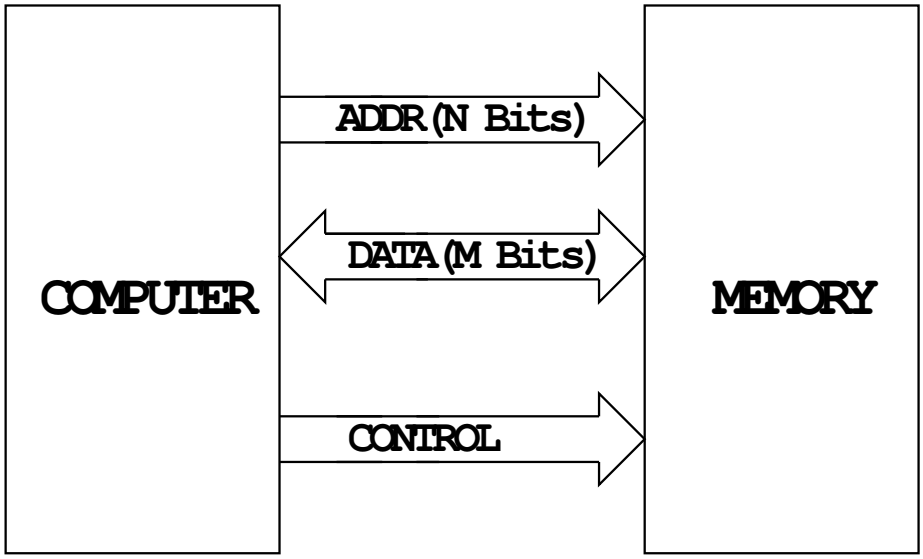
```
  msg[0] = ((year>>4)&0x0f) + '0';
msg[1] = ((year)&0x0f) + '0';
msg[2] = '/';
...
msg[8] = 0;
```

Lab on IIC Bus

- When receiving multiple bytes from slave, need to send NACK after last byte in order to tell slave to release bus.
  - If you don't do this, slave will hold onto bus, and you cannot take over bus for next operation
- Look at the flow chart on Page 39 of the IIC manual to see what to do
- I have three receive functions:
  1. `iic_receive()`: Used for receiving all but last two bytes
     - Waits for IBIF flag to set, indicating new data
     - Clears IBIf after it has been set
     - Reads data from IBDR, which starts next read
  2. `iic_receive_m1()`: Used for receiving next to last byte
     - Does same thing as `iic_receive()`, except before reading data from IBDR, it sets the TXAK bit so there will be no ACK sent on reading the last byte
  3. `iic_receive_last()`: Used for receiving last byte
     - Waits for IBIF flag to set, indicating new data
     - Clears IBIf after it has been set
     - Clears TXAK bit so ACK is re-enabled
     - Clears MS/$\overline{\text{SL}}$ bit to generate a STOP bit
     - Sets Tx/$\overline{\text{Rx}}$ bit so MC9S12 will not start SCLK to receive another byte after reading from IBDR.
     - Reads data from IBDR


PRINCETON (VON NEUMAN) ARCHITECTURE
MICROPROCESSOR
CONTROL & ADDR
CLOCK
CONTROL
MEMORY
INSTRUCTION
CONTROL
STATUS
ALU
DATA
IN
OUT


Control
Processor
X_Load
A_Load
PC_Inc
PC_Load
MAR_Load
IR_Load
Inst
Data_Mux_Sel
Addr_Mux_Sel
ALU_Ctrl
Z_Load
C_Load
V_Load
N_Load
Z
C
V
N
Mem_W
0xFF
Reset Vector
X
A
PC
MAR
IR
Data_Mux
Addr_Mux
ALU
Z
C
V
N
Address
Memory
Output
Input

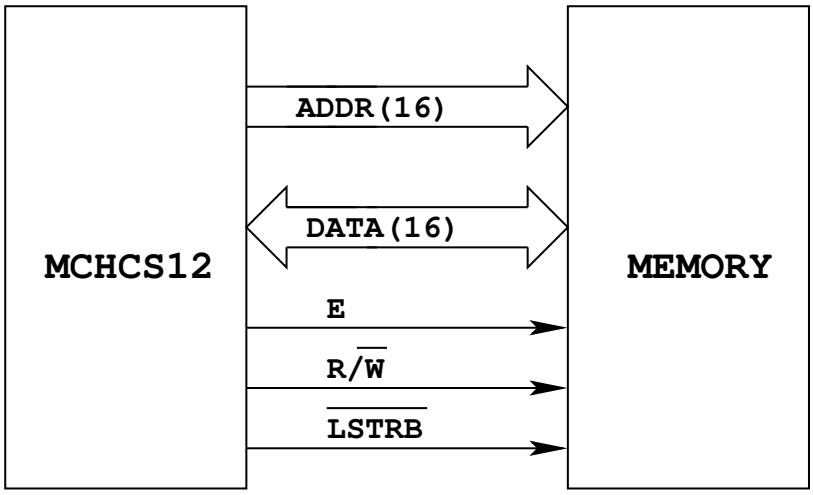


Computer with N bit address bus can access
2^N bytes of data

Computer with M bit data bus can access
M bits of data in one memory cycle

Value on address bus tells memory which
location computer wants to read (write)

Control lines tell memory when computer wants
to read (write) data, and if access is read or write



MCHCS12 has 16 bit address bus - can access 65536 bytes

1024 bytes = 1 kB

65536 bytes = 64 kB

MCHCS12 has 16 bit data bus - can access 16 bits (2 bytes) at a time

For example, the instruction LDX $0900
will read the two bytes at address $0900 and $0901

Sometimes MC9S12 only accesses one byte -- e.g., LDAA $0900
The MC9S12 accesses only the byte at address $0900

R/$\overline{W}$ tells memory if MC9S12 is reading or writing

R/$\overline{W}$ high => read
R/$\overline{W}$ low => write

E tells memory when MC9S12 is reading (writing) -- synchronizes data accesses

$\overline{LSTRB}$ tells memory if MC9S12 accessing one or two bytes

Address, Data and Control Buses

- A microprocessor system uses address, data and control buses to communicate with external memory and memory-mapped peripherals
- The address bus determines which memory location to access
- The control bus specifies whether the memory cycle is a read (into microprocessor) or a write (out of microprocessor) cycle, and specifies timing information for the cycle
- The data bus contains the data being transfered during the memory cycle
- For example, consider the following simple 9S12 program, which continuously increments the contents of address `0x0400`:

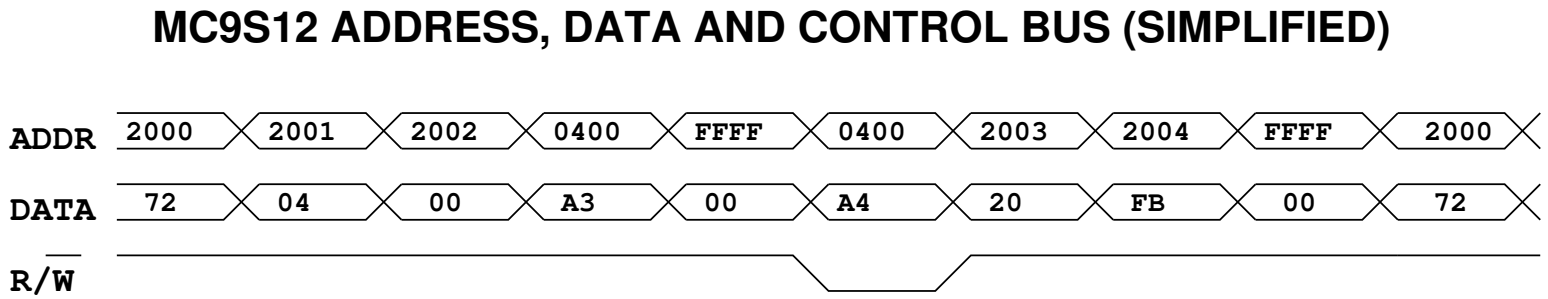
```
        org     0x2000

loop:   inc     0x0400
        bra     loop
```

  - The program is stored in memory starting at memory location `0x2000`
  - The 9S12 Program Counter starts at address `0x2000`
  - The 9S12 reads the first instruction, `inc 0x0400`, located in address `0x2000` through `0x2002`
  - The 9S12 then reads the contents of memory location `0x0400`, takes an internal memory cycle to increment the value, then writes the new value out to address `0x0400`
  - The 9S12 then reads the next instruction, `bra 0x2000`
  - The 9S12 takes one memroy cycle to load the program counter with the new value of `0x2000`, and to clear its internal pipeline, then reads the instruction at `0x2000` to figure out what to do next

The 9S12 address, data and control buses (simplified)

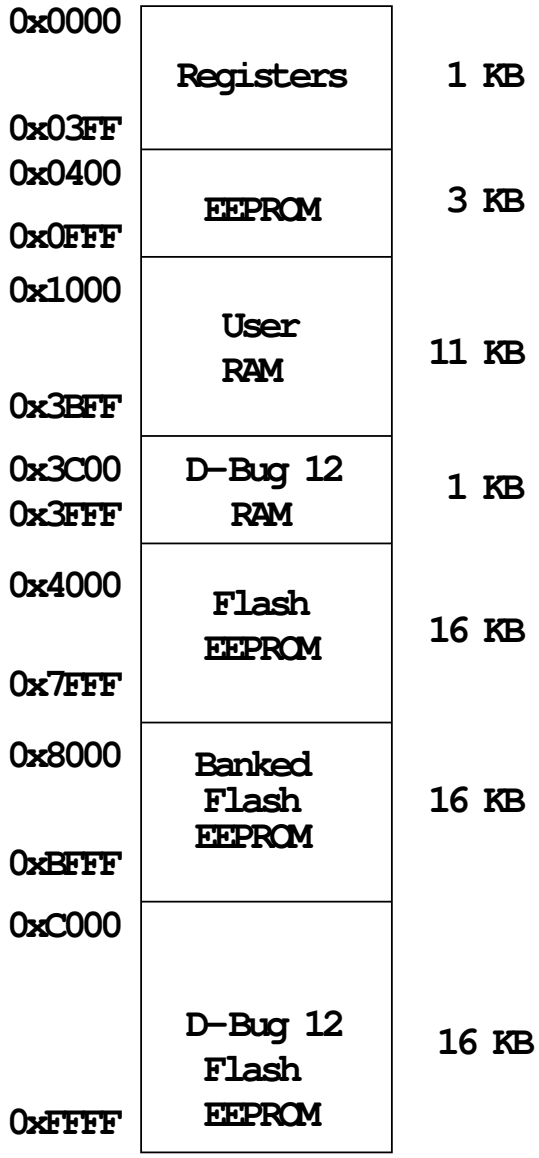
- Note: The following diagram assumes that the 9S12 accesses one byte at a time
- The 9S12 actually accesses two bytes (16 bits) at a time, when it can
- What actually occurs on the 9S12 bus is a little more complicated than what is shown below




```
                .org   0x2000        2000:  72
                                     2001:  04   inc 0x0400
loop:           inc    0x0400        2002:  00
                                     2003:  20   bra 0x2000
                bra    loop          2004:  FB
```

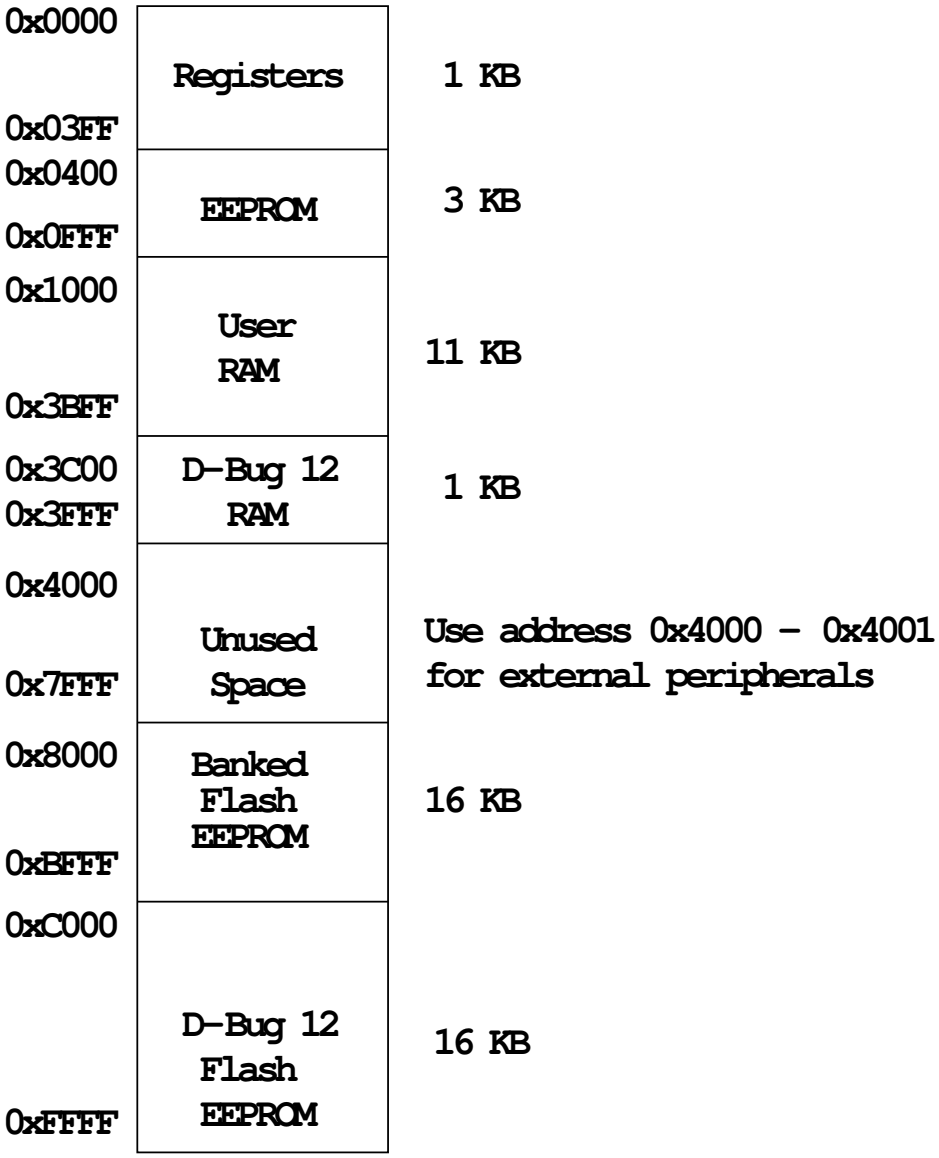
The 9S12 Memory Map

- The 9S12 has address regions occupied by internal memory and peripherals
- A diagram showing which address regions are used is called a memory map
- Here is a memory map of the 9S12DP256 with no added memory or peripherals



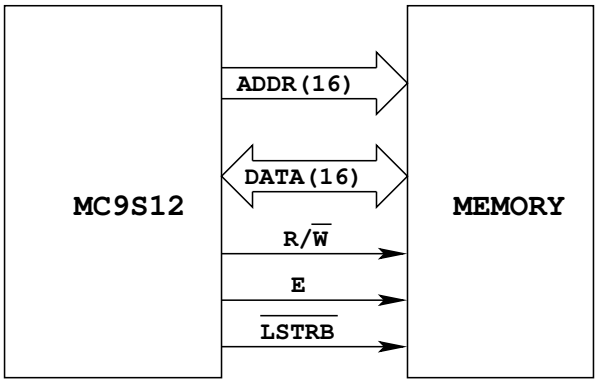
The Expanded 9S12 Memory Map

- We will add external peripherals to the 9S12
- Here is a memory map of the MC9S12DP256 with the peripherals we will add
- The peripherals will be put at 0x4054 and 0x4055



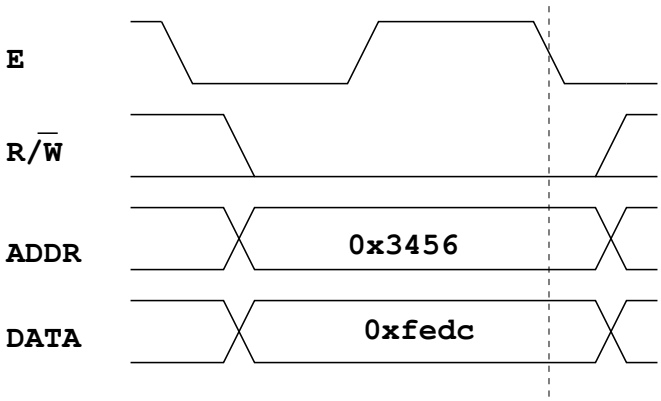
Simplified 9S12 Write Cycle

- When the 9S12 writes data to memory it does the following:
  - It puts the address it wants to write to on the address bus (when E-clock goes low)
  - It puts the data it wants to write onto the data bus
  - It brings the Read/Write (R/$\overline{W}$) line low to indicate a write
  - The 9S12 expects the external device at the given address will latch the data into its registers data on the falling edge of the E-clock



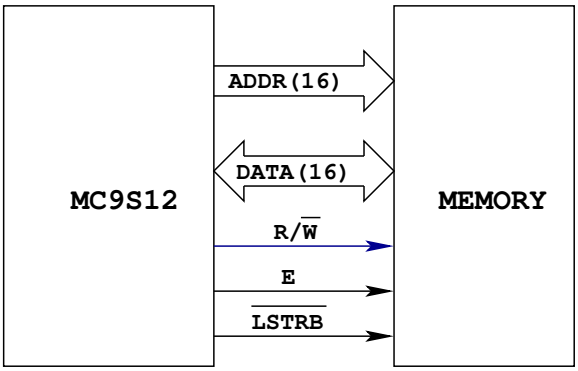
WRITE: MC9S12 puts address on address bus
puts data on data bus
brings R/W low
Memory latches data on falling edge of E clock

Example: Write 0xfedc to address 0x3456 & 3457



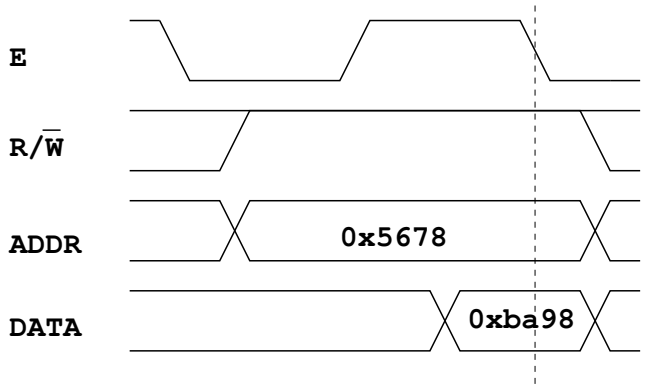
Simplified 9S12 Read Cycle

- When the 9S12 reads data from memory it does the following:
  - It puts the address it wants to read from on the address bus (when E-clock goes low)
  - It brings the Read/Write (R/$\overline{\text{W}}$) line high to indicate a read
  - The 9S12 expects the external device at the given address will put data on the data bus
  - On the falling edge of the E-clock, the 9S12 latches the data into its internal registerl



READ: MC9S12 puts address on address bus
brings R/W high
Memory puts data on data bus
HC12 latches data on falling edge of E clock

Example: Read from address 0x5678 & 0x5679



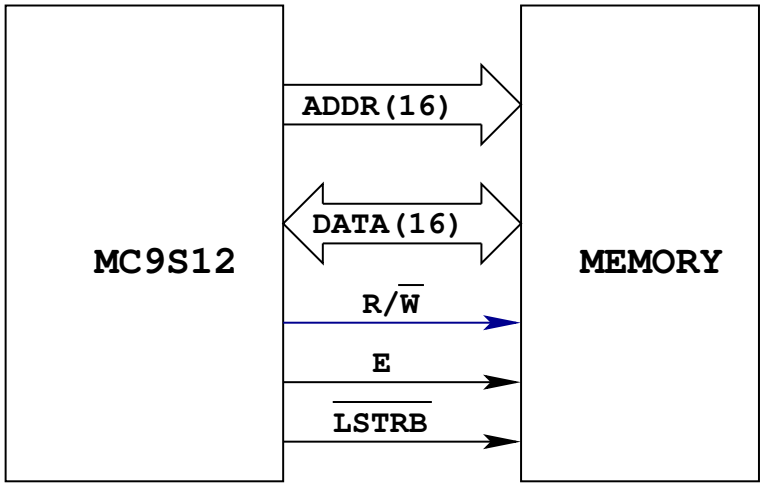
The Real MC9S12DP256 Bus

- Up to now we have been using the 9S12 in Single Chip Mode
  - In Single Chip Mode the 9S12 does not have an external address/data bus
- The 9S12 can be run in Expanded Mode
  - In Expanded Mode the 9S12 does have an external address/data bus
- Things are a little more complicated on the real MC9S12DP256 bus than shown in the simplified diagrams above
- The MC9S12DP256 has a multiplexed address/data bus
- The MC9S12DP256 sometimes accesses a single byte on a memory cycle, and it sometimes access two bytes on a memory cycle

The Multiplexed Address/Data Bus

- The MC9S12DP256 has a limited number of pins it can use
- To have full 16-bit address bus and a full 16-bit data bus the MC9S12DP256 would need to use 32 extra pins (in addition to several pins used for the control bus)
- To save pin count Motorola uses the same set of pins for several purposes
- When put into expanded mode, the 9S12 uses the pins normally used for Ports A and B for its mulitplexed address and data bus
  - When running in expanded mode you can no longer use Ports A and B as general purpose I/O lines
- The 9S12 uses the same sixteen line of Ports A and B for both address and data
- When the E-clock is low the sixteen lines AD15-0 are used for address
- When the E-clock is high the sixteen lines AD15-0 are used for data

The Multiplexed Address/Data Bus



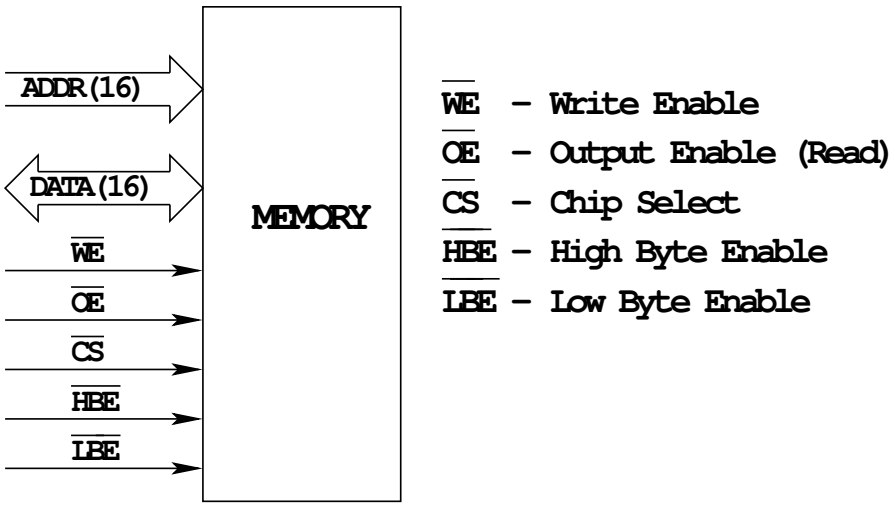
MC9S12 has 16-bit address and 16-bit data buses

Requires 35 bits

Not enough pins on MC9S12 to allocate 35 pins for buses and pins for all other functions

Memory Chip Interface

- Memory chips need separate address and data bus
  - Need way to de-multiplex address and data lines from 9S12
- Memory chips need different control lines than the 9S12 supplies
- These control lines are:
  - Chip Select – goes low when the 9S12 is accessing memory chip
  - Write Enable – goes low when the 9S12 is writing to memory
  - Output Enable – goes low when the 9S12 is reading from memory
  - High Byte Enable – goes low when the 9S12 is accessing the High Byte (Odd Address) of memory
  - Low Byte Enable – goes low when the 9S12 is accessing the Low Byte (Even Address) of memory

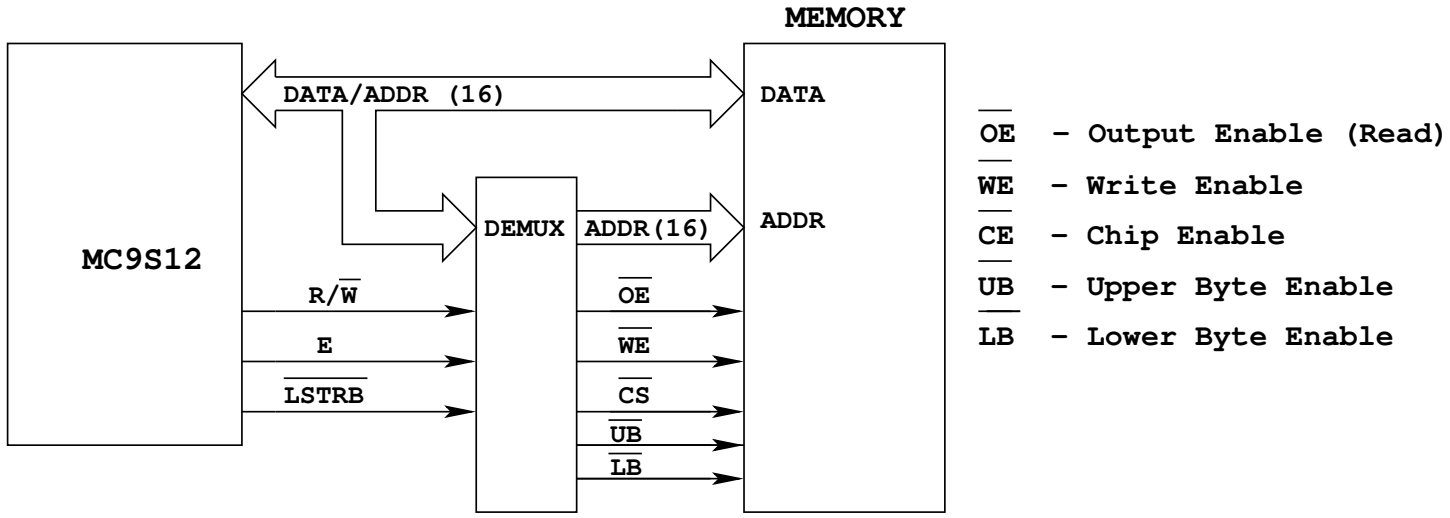


Memory needs separate address and data busses

Need way to separate address and data

The Multiplexed Address/Data Bus

- To talk to memory chip we will need to build a demultiplexer between the 9S12 and the memory chip



MC9S12 has 16-bit address and 16-bit data buses

Requires 35 bits

Not enough pins on MC9S12 to allocate 35 pins
for buses and pins for all other functions

Solution: multiplex address and data buses

16-bit Bus: While E low, bus supplies address
While E high, bus supplies data