Exam III Review

ECE 3710

be calm...

So far, this is the oldest

I've been.

- George Carlin

pointers:

1. if something is said to be configured in the problem statement, don't waste time configuring it
2. if something isn't configured, you must configure it

(this is time consuming so you pick least difficult thing to configure)

I'm testing to see if you know that thing

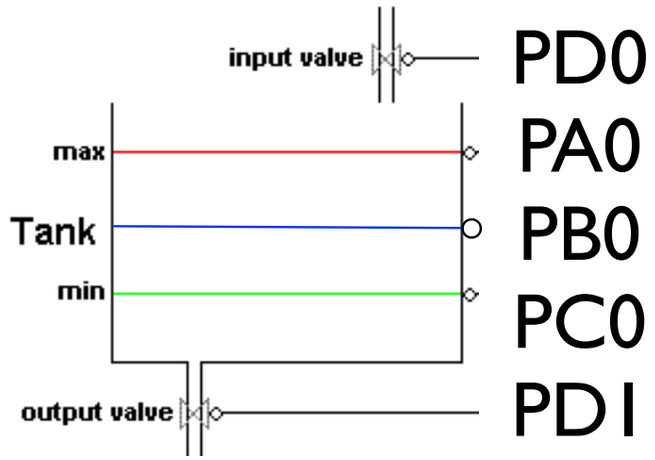
exam two

1. a. PA0 h2l; PB0 h2l; PC0 l2h

You are to write an assembly program that uses interrupts to maintain the level of liquid in a tank. If the tank is too full (red line) the program should open a valve that releases liquid until the level reaches the target level (blue line); if it's not full enough (green line) it should open another valve that adds liquid until the level reaches the target level. Three sensors are used to determine the level of the liquid. For red and green sensors, if the liquid is above the level the sensor outputs a zero, otherwise it outputs one. If the liquid is at the target level the blue sensor outputs a zero, otherwise a one. A logic high closes a valve and a logic low opens it.

(a) (5 points) Indicate which pins the sensors and valves are connected to on the figure below and **state** how the uC is configured to handle GPIO external interrupts.

(b) (20 points) Assuming the above configuration, write the ISR(s). You may use polling in your ISR(s).



don't have to check to see which caused IRQ

exam two

1. a. PA0 h2l; PB0 h2l; PC0 l2h

```
GPIOA_Handler ;too much water
ldr R1,=PA
mov R0,#0x1
str R0,[R1,0x41C] ;ack IRQ
ldr R1,=PD
mov R0,#0x1
str R0,[R1,0x3FC] ;open bottom value
bx LR
```

```
GPIOB_Handler ;close valves
ldr R1,=PB
mov R0,#0x1
str R0,[R1,0x41C] ;ack IRQ
; close all valves
ldr R1,=PD
mov R0,#0x3
str R0,[R1,0x3FC]
bx LR
```

exam two

II. assume char ptr to PRI0 named PRI0

get priority (have to shift)
check for increase or decrease
increase or decrease (don't forget bounds)
save back

Your system uses interrupts zero through three (vector number 16–19); each time an ISR is run a counter is incremented for it. To prevent one interrupt source from taking over the system you will adjust the priority of the ISRs based on how frequently they are run. If an ISR is run more than ten times per 100 ms it's priority should be decreased by one; if it's run less than ten times it should be increased by one. A R/W array that keeps track of the number of times the ISR has been run is accessible by the pointer `unsigned short *CNT`. Assume that `SysTick` has been configured to issue an IRQ every 100 ms. Write the `SysTick` ISR in C to accomplish the above.

exam two

```
SysTick_Handler
{
  unsigned char PRI;
  for(int i =0; i<4; i++)
  {
    PRI = PRI0[i] >> 5; //get current priority
    if(CNT[i] > 10 && PRI < 7); //dec if not lowest already
      PRI0[i] = (PRI+1) << 5;
    else if(CNT[i] < 10 && PRI > 0); //inc if not highest already
      PRI0[i] = (PRI-1) << 5;
    CNT[i] = 0; //clear cnt
  }
```

exam two

III. IRQ on RX, >= 7/8 full; UART0; char. ptr to UART0 base; SysTick stopped; int var for STCTRL

don't overwrite config for systick

You have a uC that performs calculations on data received over a serial interface. Since data comes in faster than the uC can process it the data has to be buffered; you have enabled the RX FIFO buffer. Unfortunately, sometimes the buffer fills up so the transmitter needs to be told to stop sending data, which is done by sending `0x13` via the serial port. When the uC is ready for more data, transmission is re-enabled by sending `0x11`.

In C write a serial ISR that tells the transmitter to stop sending data when the RX buffer is full. After this you should check to see if there is any space in the buffer every one ms and if there is alert the transmitter that it may start sending data again. Assume `SysTick` has been configured to expire every one ms.

(a) (5 points) State which events cause the uC to issue an IRQ.

(b) (20 points) Using multiple ISRs, if necessary, implement the above.

exam two

```
UART0_Handler
{
  UART0[0x44] = 0x10; //ack irq
  if((UART0[0x18]&0x40)==0x40) //check if RX buffer is full RXFF=1
  {
    UART0[0x0] = 0x13; //send stop
    STCTRL |=0x01; //start systick
  }
```

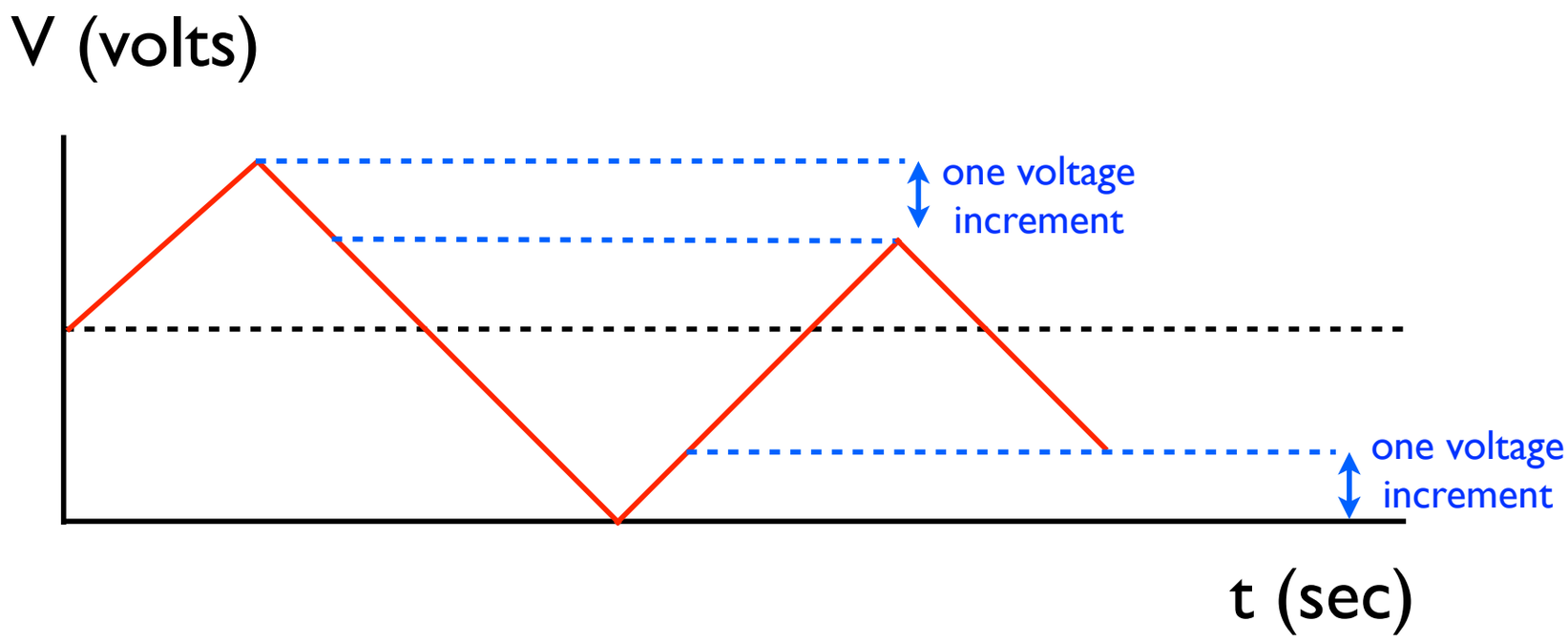
```
SysTick_Handler
{
  if((UART0[0x18]&0x40)==0) //check if buffer still full RXFF=1
  {
    UART0[0x0] = 0x11; //send start byte
    STCTRL &=0xFE; //disable systick
  }
}
```

exam two

IV.have to start/stop at middle code (127); decrement max value each time we start

one voltage increment = one code

Write a function in C that uses the DAC to create a decaying triangle wave. For each subsequent period the wave amplitude should decrease by one voltage increment (see figure below). Assume an 8-bit DAC and that the uC is configured so that the DAC will output whatever code is put in the register referred to by the global variable `DAC_CR`. Use each DAC code and repeat the waveform.

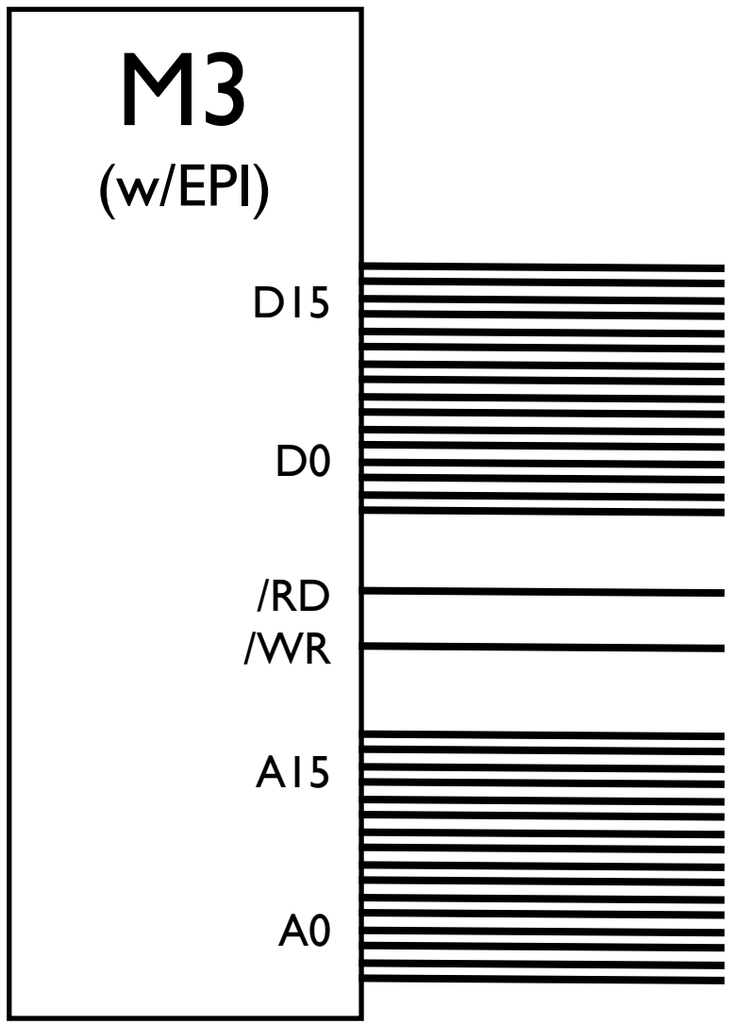


exam two

```
while(1) {
  for(int i = 256;i>0;i--) //control high/low stopping point
  {
  for(int j = 127;j<i;j++) //count up from middle
    DAC_CR = j;
  for(int j = i;j>256-i;j--) //count down from top to bottom
    DAC_CR = j;
  for(int j = 256-i;j<127;j++) //count from bottom to middle
    DAC_CR = j;
  }
}
```

R0 = (R0+1)%256 ⟶

```
add R0,#1 ;update value
and R0,#0xFF
```



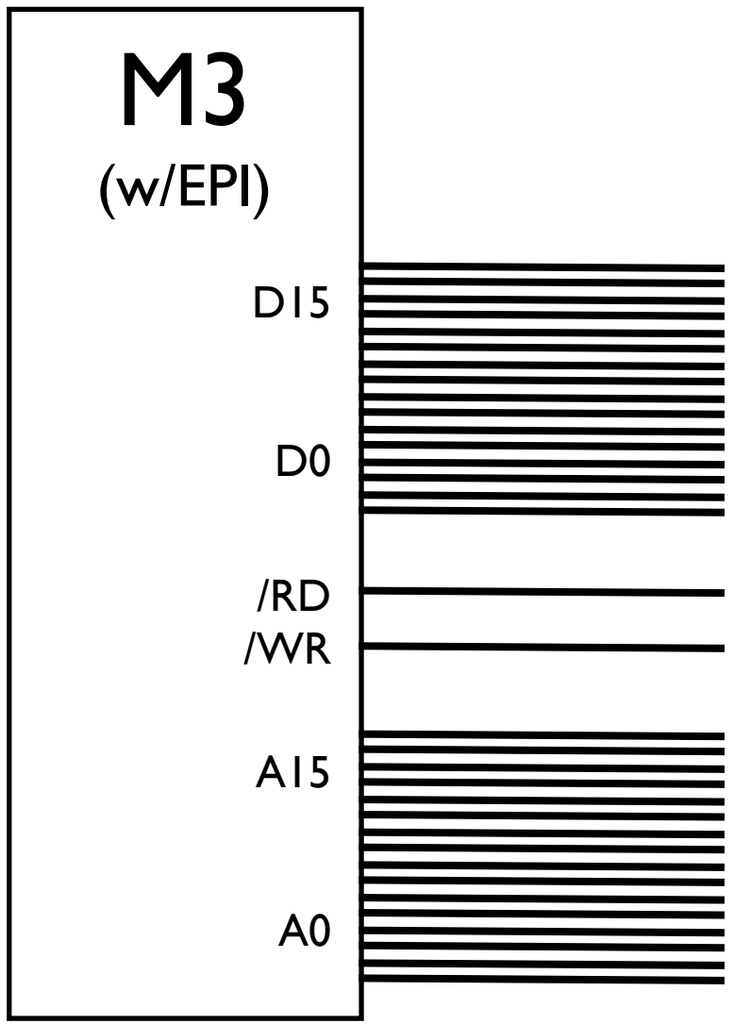
assume:

1. EPI operating in demultiplexed mode
2. EPI supports 16-bit addresses and data
3. base external memory mapped addr is 0xA0000000
4. device is byte addressable

(a) (5 points) Using the diagram on the next page, show how you would connect four `8Kx16` devices to an 8051 variant that uses a shared, de-multiplexed bus. In addition to address and data pins, each device has the following pins: `/CS, /RD, /WR`.

(b) (5 points) Provide a memory map for the configuration.

(c) (15 points) Write code in C that copies an array of length `N` from address `0x0AAA` of device one to address `0x1555` of device three.



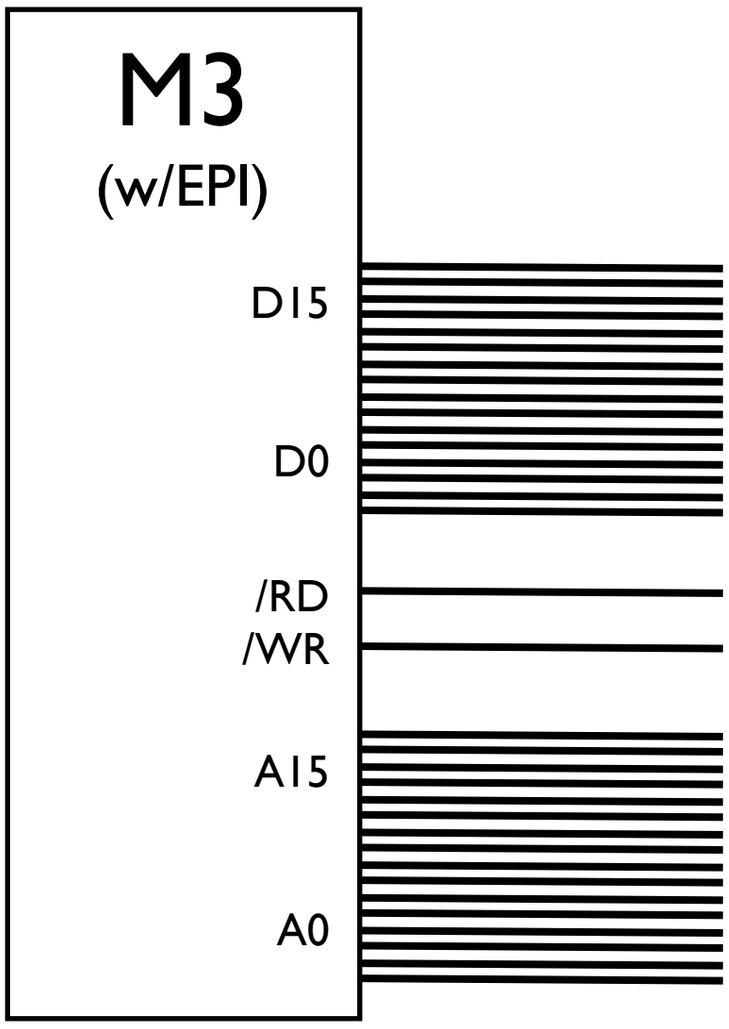
assume:

1. EPI operating in demultiplexed mode
2. EPI supports 16-bit addresses and data
3. base external memory mapped addr is 0xA0000000
4. device is byte addressable

(a) (5 points) Using the diagram on the next page, show how you would connect four `8Kx16` devices to an 8051 variant that uses a shared, de-multiplexed bus. In addition to address and data pins, each device has the following pins: `/CS, /RD, /WR`.

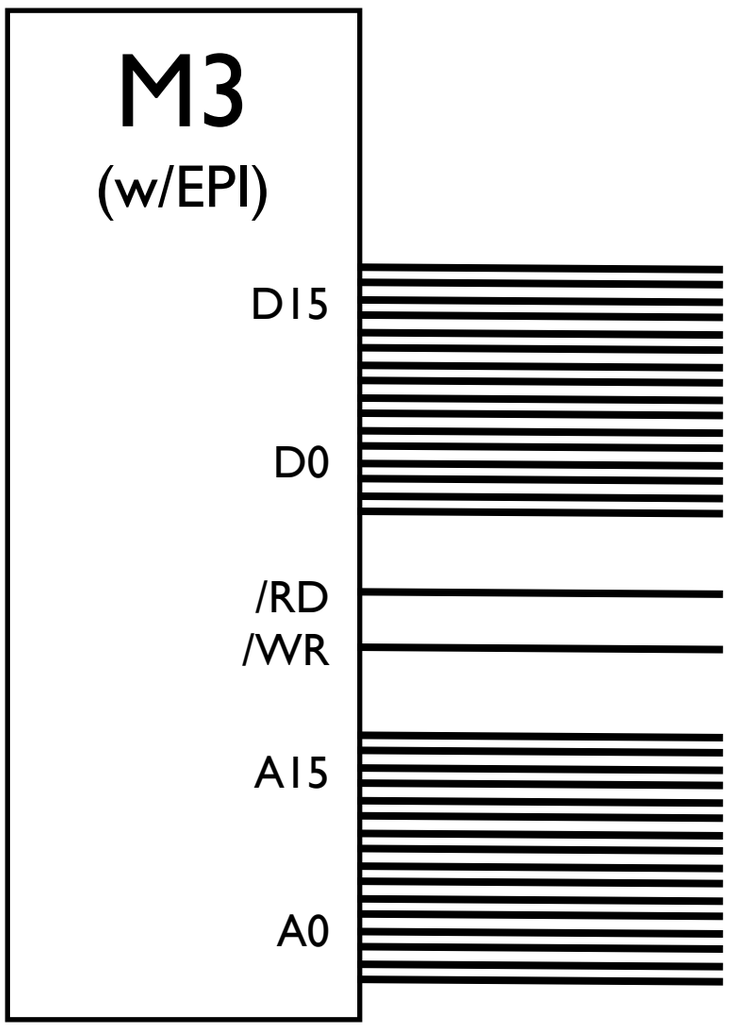
(b) (5 points) Provide a memory map for the configuration.

(c) (15 points) Write code in C that copies an array of length `N` from address `0x0AAA` of device one to address `0x1555` of device three.



Q1: how many address pins does each device have?

four 8Kx16 devices



Q1: how many address pins does each device have?

four 8Kx16 devices

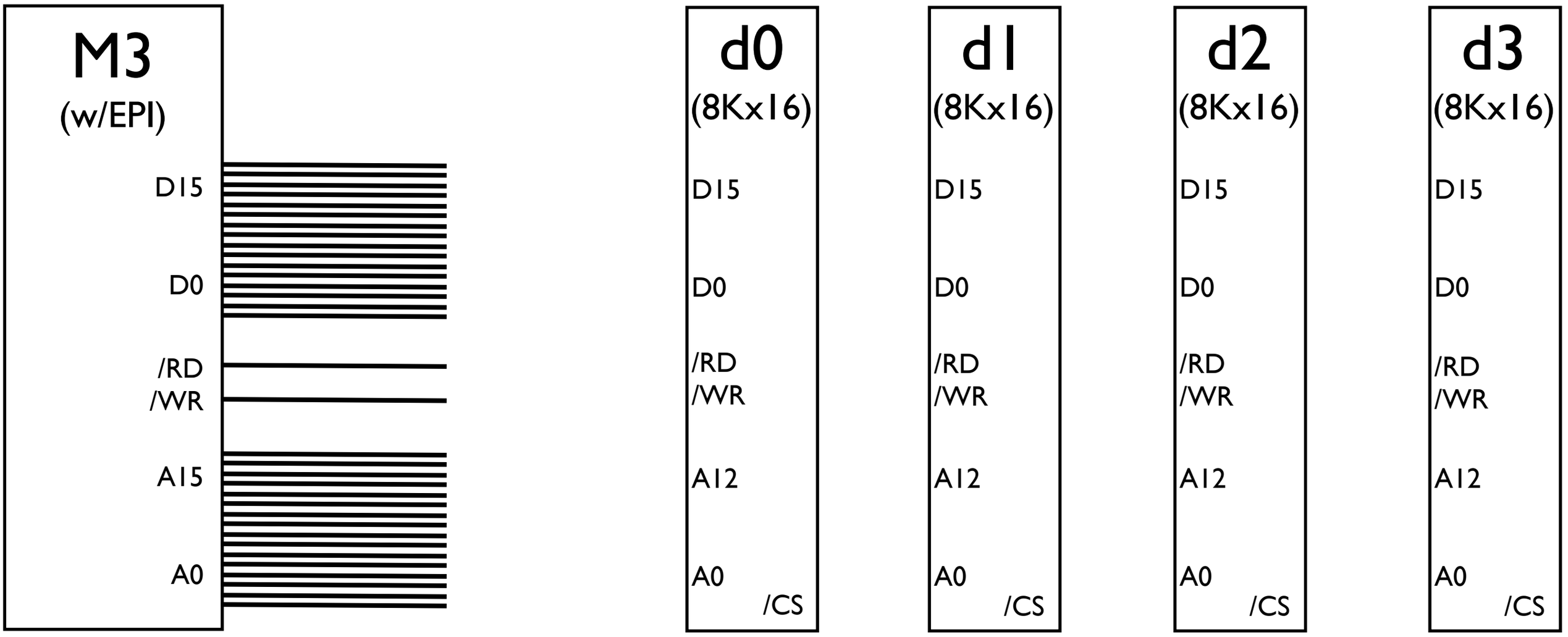
A1: 2^13 (byte addr) = 8 K (base two)

address pins = 13

note: capacity = 2^(13-1)*16 = 64 KB

Q2: how to connect devices

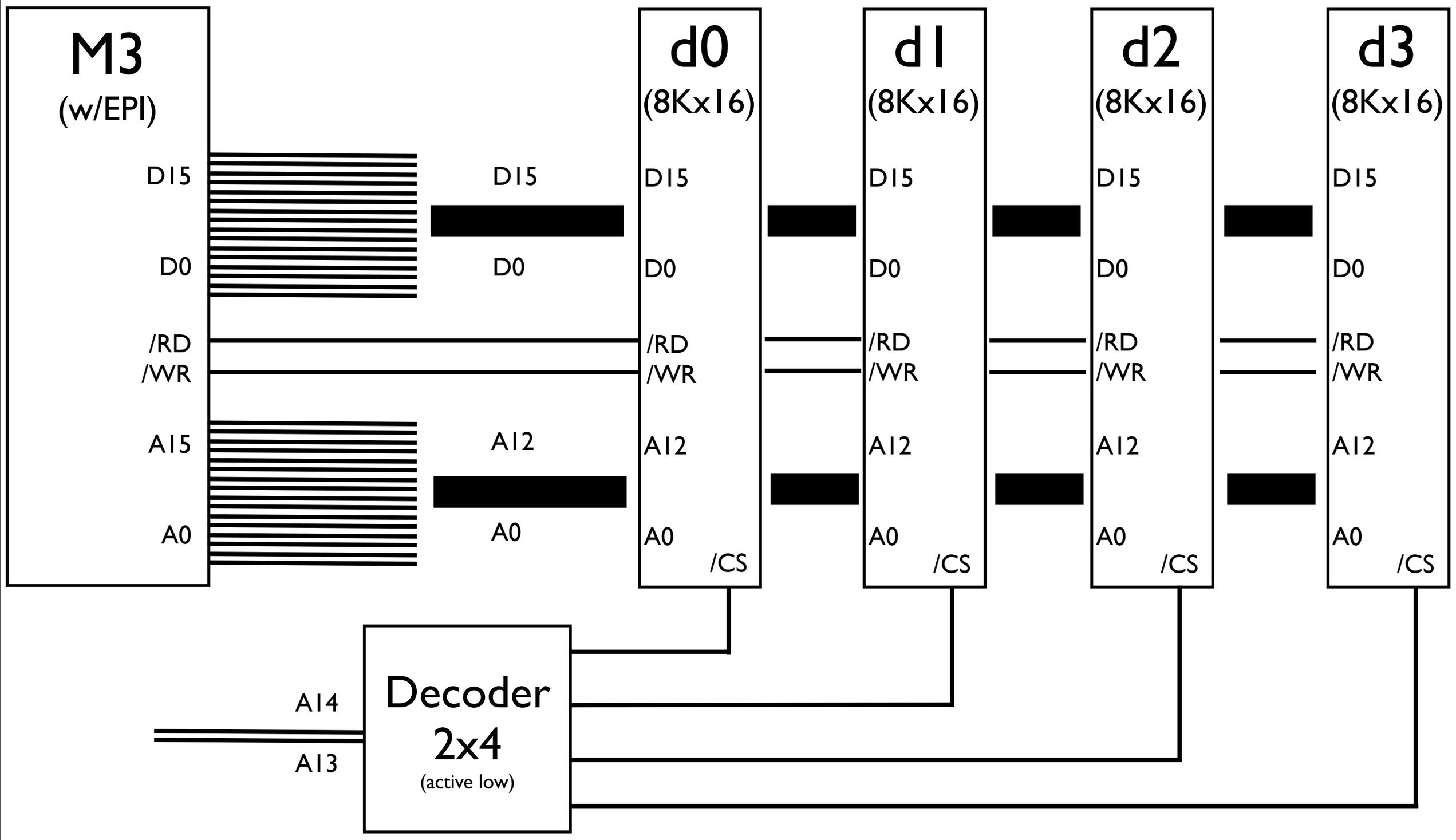
(thirteen pins for addr, three left for CS)



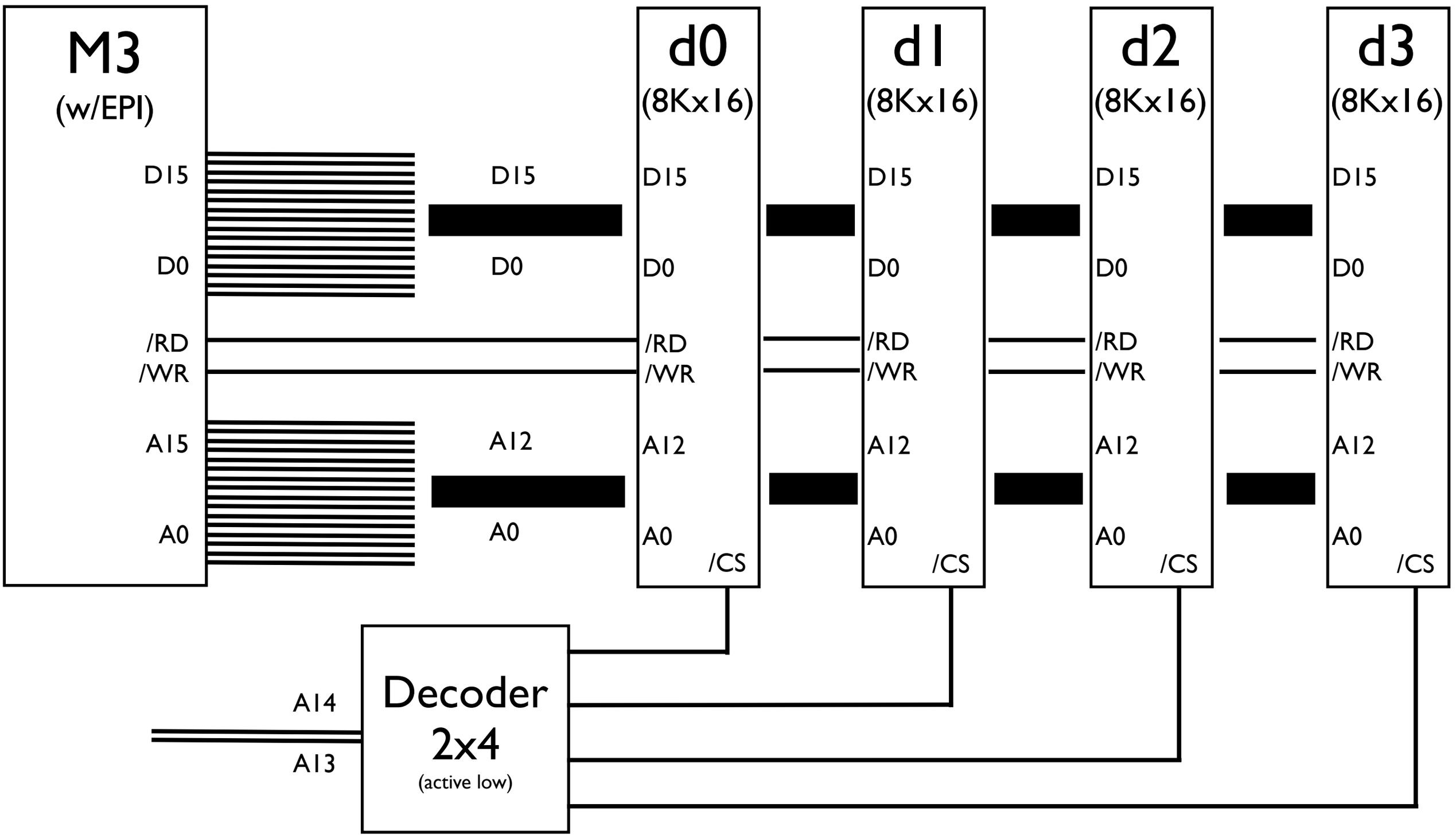
Q2: how to connect devices

(thirteen pins for addr, three left for CS)

A2: active low decoder



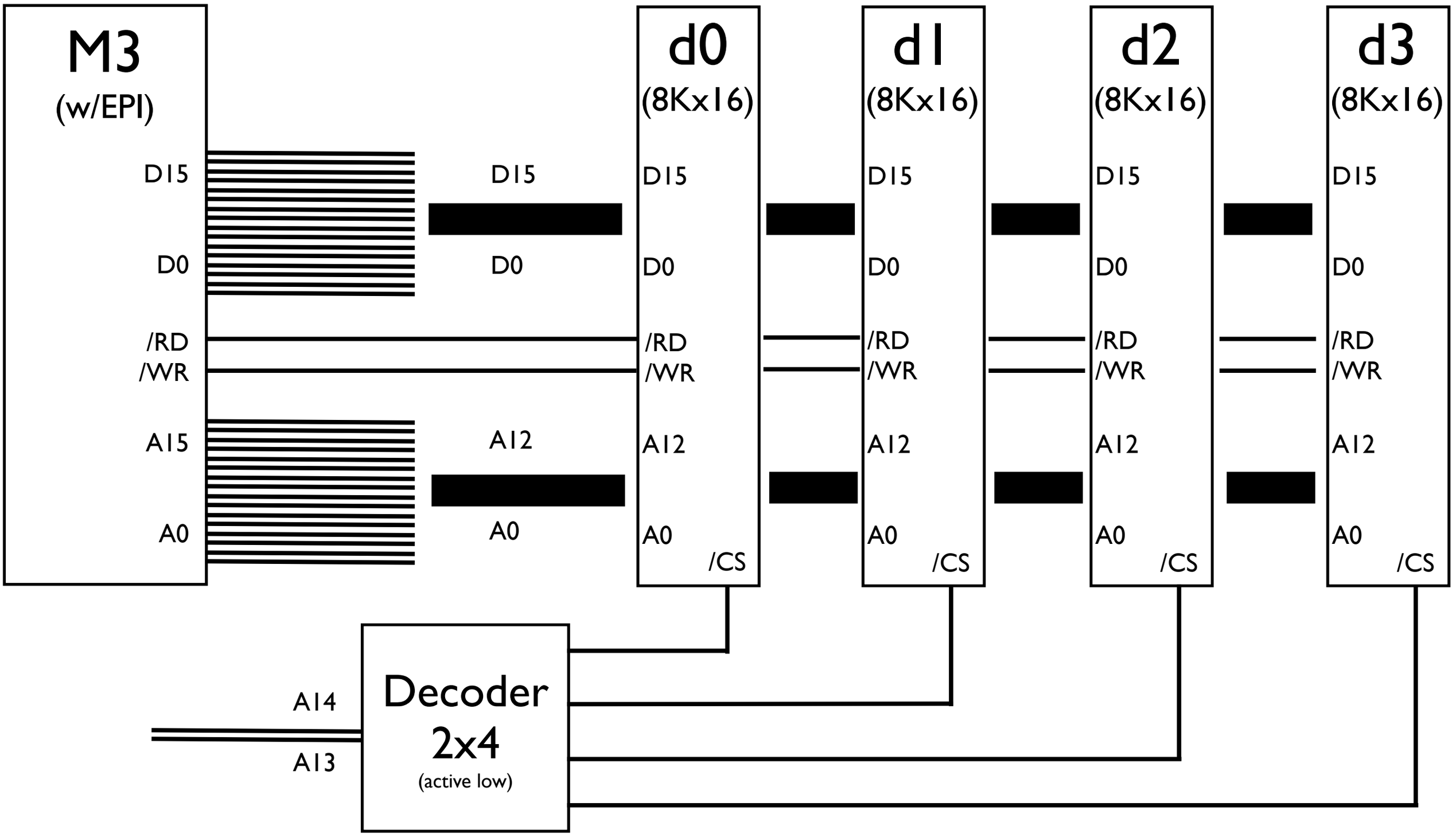
Q3: memory map?



Q3: memory map?

A3: d0: 0bX00X...X (13-bits) d2: 0bX10X...X (13-bits)

(offsets) d1: 0bX01X...X d3: 0bX11X...X



Q3: memory map?

A3: d0: 0bX00X...X (13-bits) d2: 0bX10X...X (13-bits)

(offsets) d1: 0bX01X...X d3: 0bX11X...X

	min offset	max offset
d0:	0bX000...0	0bX001...1
d1:	0bX010...0	0bX011...1
d2:	0bX100...0	0bX101...1
d3:	0bX110...0	0bX111...1

	min offset	max offset
d0:	0x0000	0x1FFF
d1:	0x2000	0x3FFF
d2:	0x4000	0x5FFF
d3:	0x6000	0x7FFF

why did we remove don't cares?

Q3: memory map?

A3:

	min offset	max offset
d0:	0x0000	0x1FFF
d1:	0x2000	0x3FFF
d2:	0x4000	0x5FFF
d3:	0x6000	0x7FFF

↓

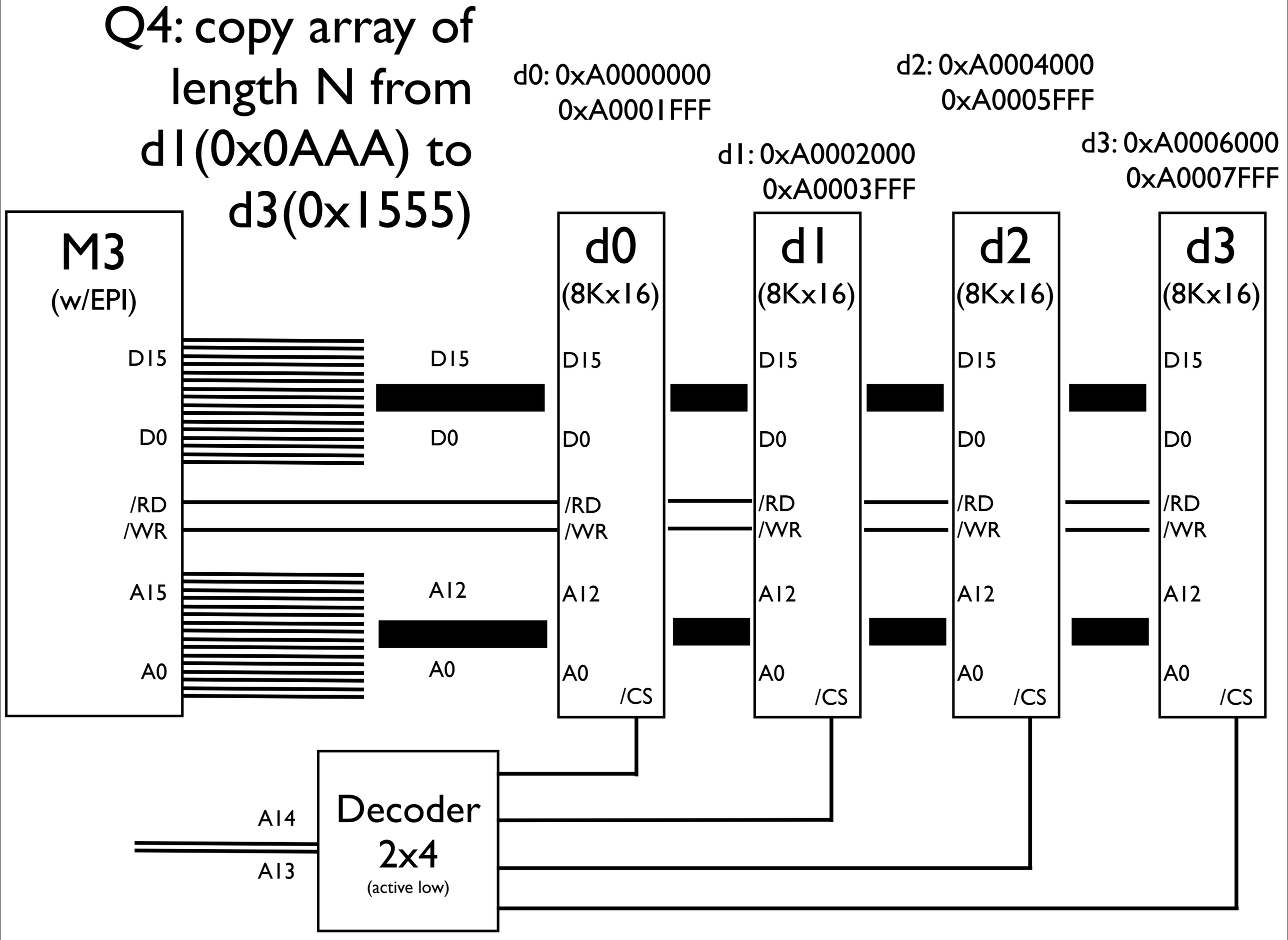
uC mem. addr

(0xA0000000 is base)

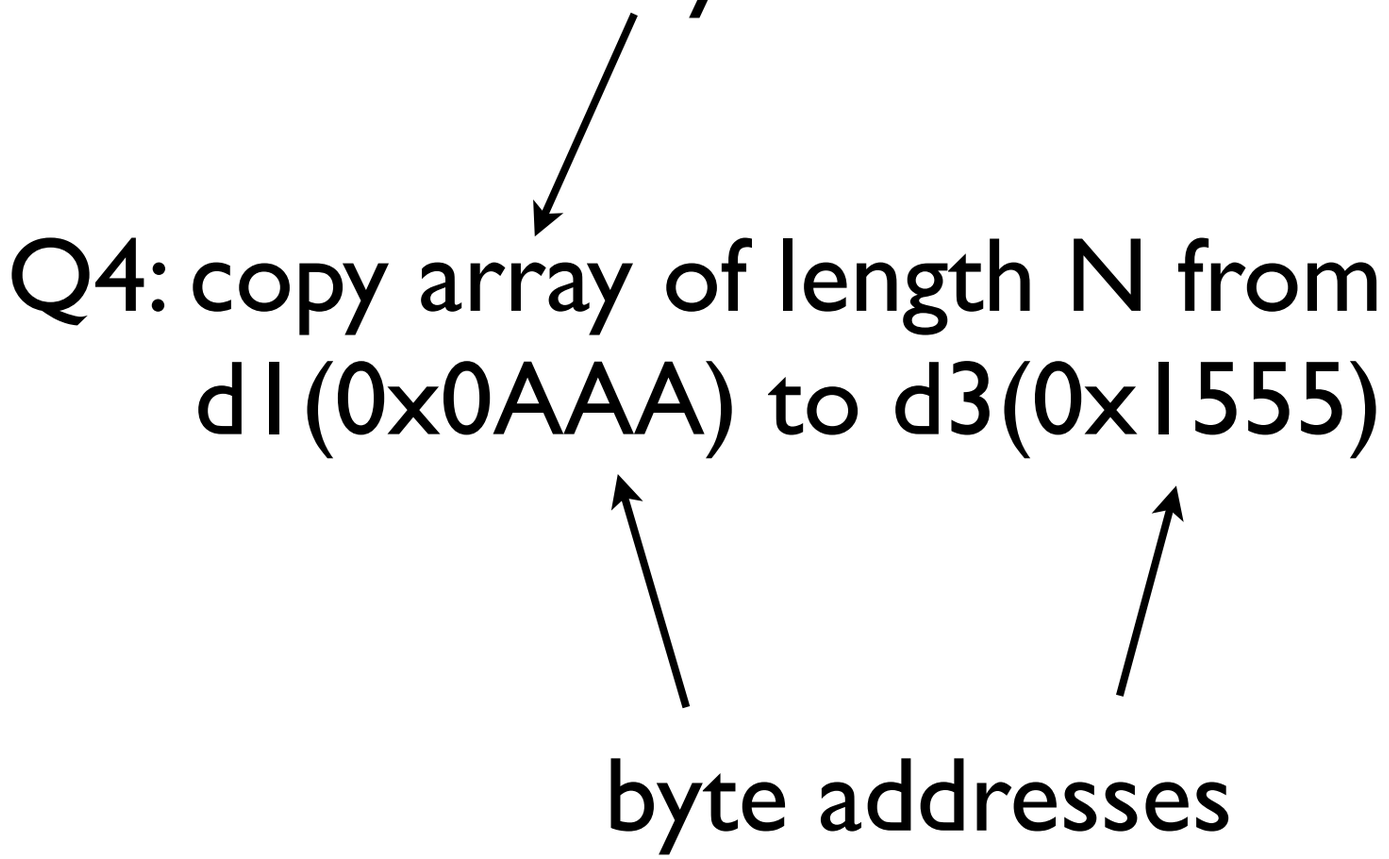
↓

d0:	0xA0000000	0xA0001FFF
d1:	0xA0002000	0xA0003FFF
d2:	0xA0004000	0xA0005FFF
d3:	0xA0006000	0xA0007FFF

Q4: copy array of length N from d1(0x0AAA) to d3(0x1555)



16-bit device: presumably
array of shorts



addressing for char vs. short arrays:

```
unsigned char *charArray = (unsigned char *) 0x0;
unsigned short *shortArray = (unsigned short *) 0x0;
```

byte addr pointed to is 0x2

```
shortArray[1] = 0x12
```

equivalent to

```
charArray[2] = 0x2
charArray[3] = 0x1
```

Q4: copy array of length N from
d1(0x0AAA) to d3(0x1555)

byte addresses

short addresses: d1(0xAAA/2) to d3(0x1555/2)

```
//D1 base addr
unsigned short *D1 = (unsigned short *) 0xA0002000;
//D3 base addr
unsigned short *D3 = (unsigned short *) 0xA0006000;

for(int i=0;i<N;i++)
  D3[0x1555/2 + i] = D1[0xAAA/2 + i];
```

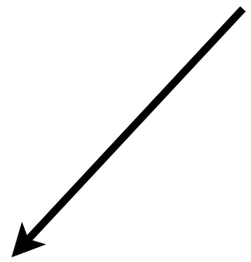
0x1555=5461 /2 2730.5

(could present problems)

d1: 0xA0002000

d3: 0xA0006000

define ptr that points to src and dst byte addresses instead using offsets (via array indexing) from base address of devices



correct way:

```
//0xA0002000 | 0x0AAA (D1 base addr + offset of src)
unsigned short *D1_src = (unsigned short *) 0xA0002AAA;
//0xA0006000 | 0x1555 (D3 base addr + offset of target)
unsigned short *D3_dst = (unsigned short *) 0xA0007555;

for(int i=0;i<N;i++)
  D3_dst[i] = D1_src[i];
```

d1: 0xA0002000

d3: 0xA0006000

George Carlin's 15 Rules to Live By:

2. Whatever it is you pursue, try to do it just well enough to remain in the middle third of the field. Keep your thoughts and ideas to yourself and don't ask questions. Remember, the squeaky wheel is the first one to be replaced.

also, remember...doom!