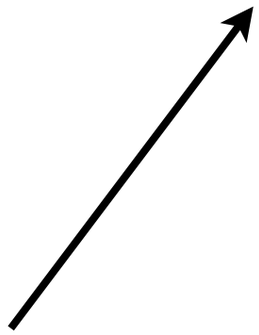
Memory-mapped External Peripherals II

ECE 3710

I have had a perfectly wonderful evening, but this wasn't it.

- Groucho Marx

external memory mapping:



1. data written to uC memory is sent to external device
2. data read from uC memory comes from external device

uC is configured to handle:

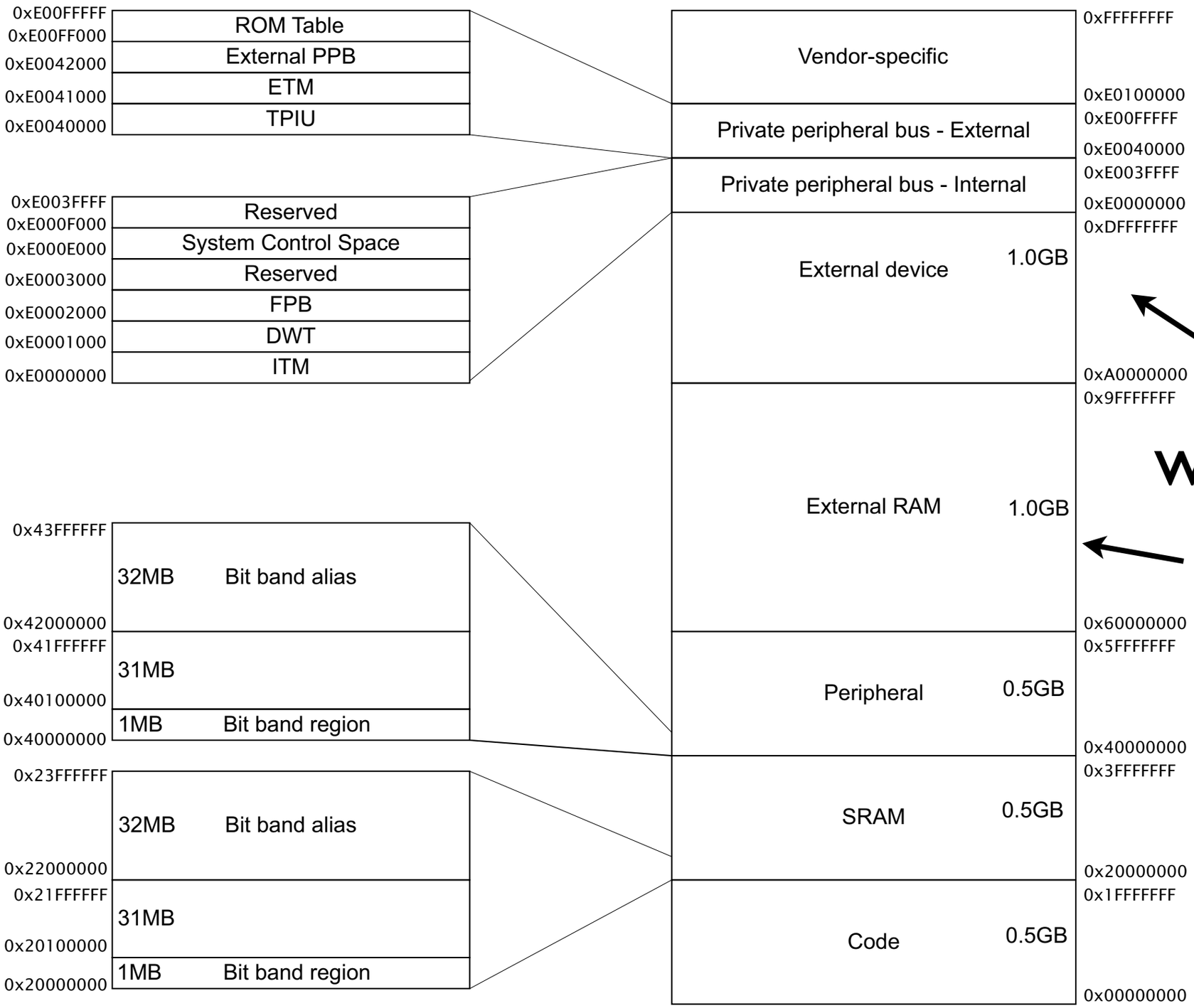
1. timing
2. pin assignments for data/addr
3. sending data/addr
4. receiving data

result: external device appears local

(reads write don't require your code to set/read pins directly)

ARM memory map

which addresses point to which things



writes here directed at external devices

addr sent on i/o pins;
data sent/recv on i/o pins

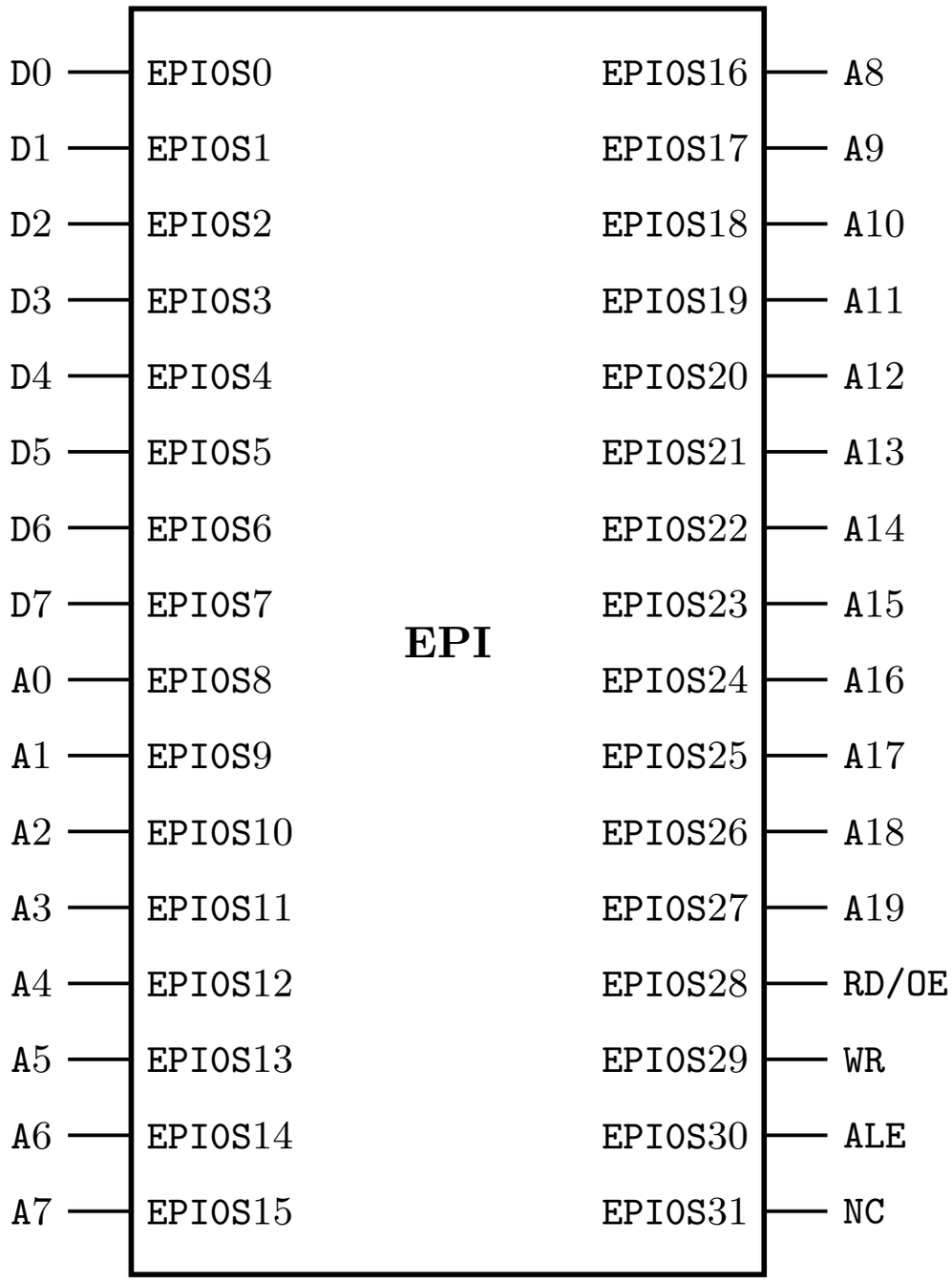
EPI: 8-bit demultiplexed mode

8-bits

address and data

2^20 of something

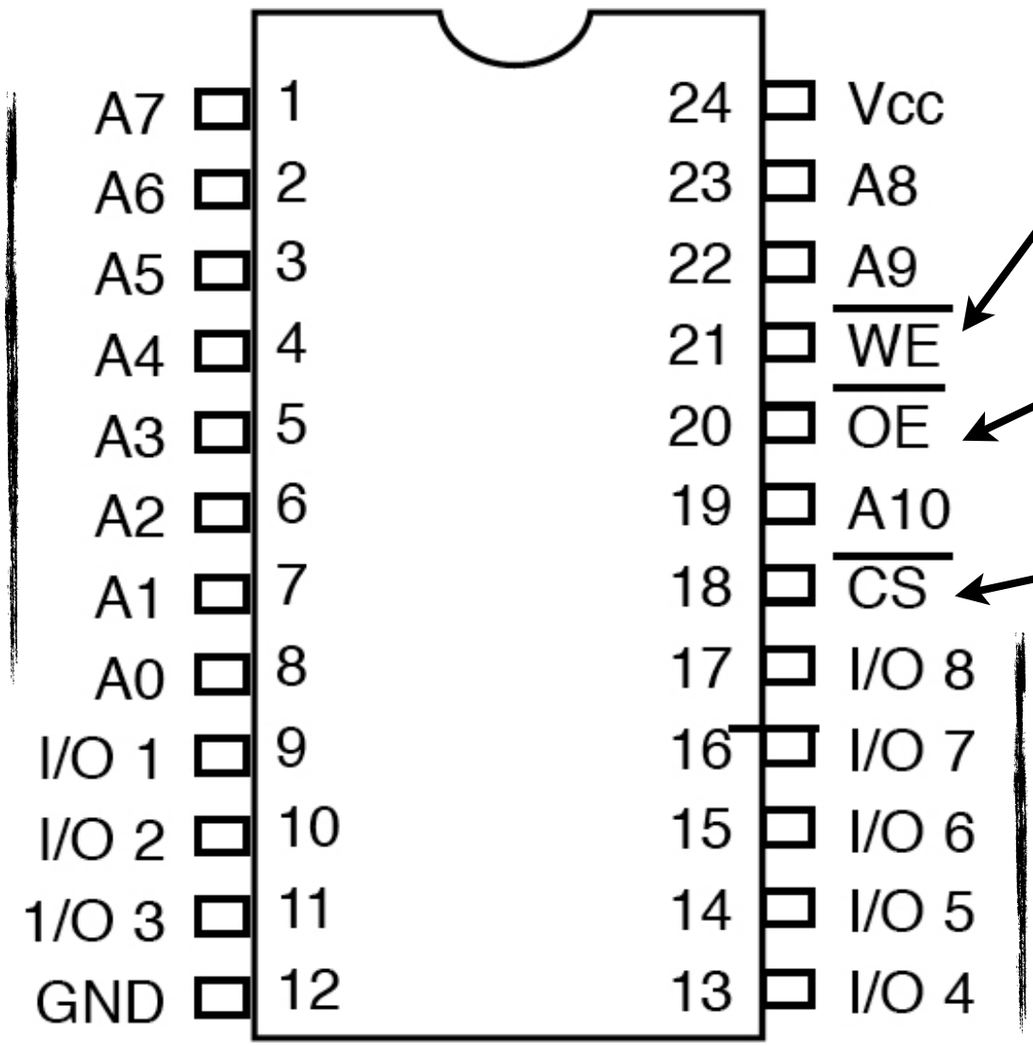
(doesn't have to be bytes)



D[7:0] = DATA[7:0] ⟵ first byte of data

A[19:0] = 0xA00[00000] ⟵ written here

example: SRAM



address
(2^8 bytes)

pins A0--7
(A8--A10 grnd)

enable writing

enable reading

activate device for i/o (sometimes CE)

data
(8 bits)

write 0xAA to address 0x34:

SRAM needs to see
(at same time)

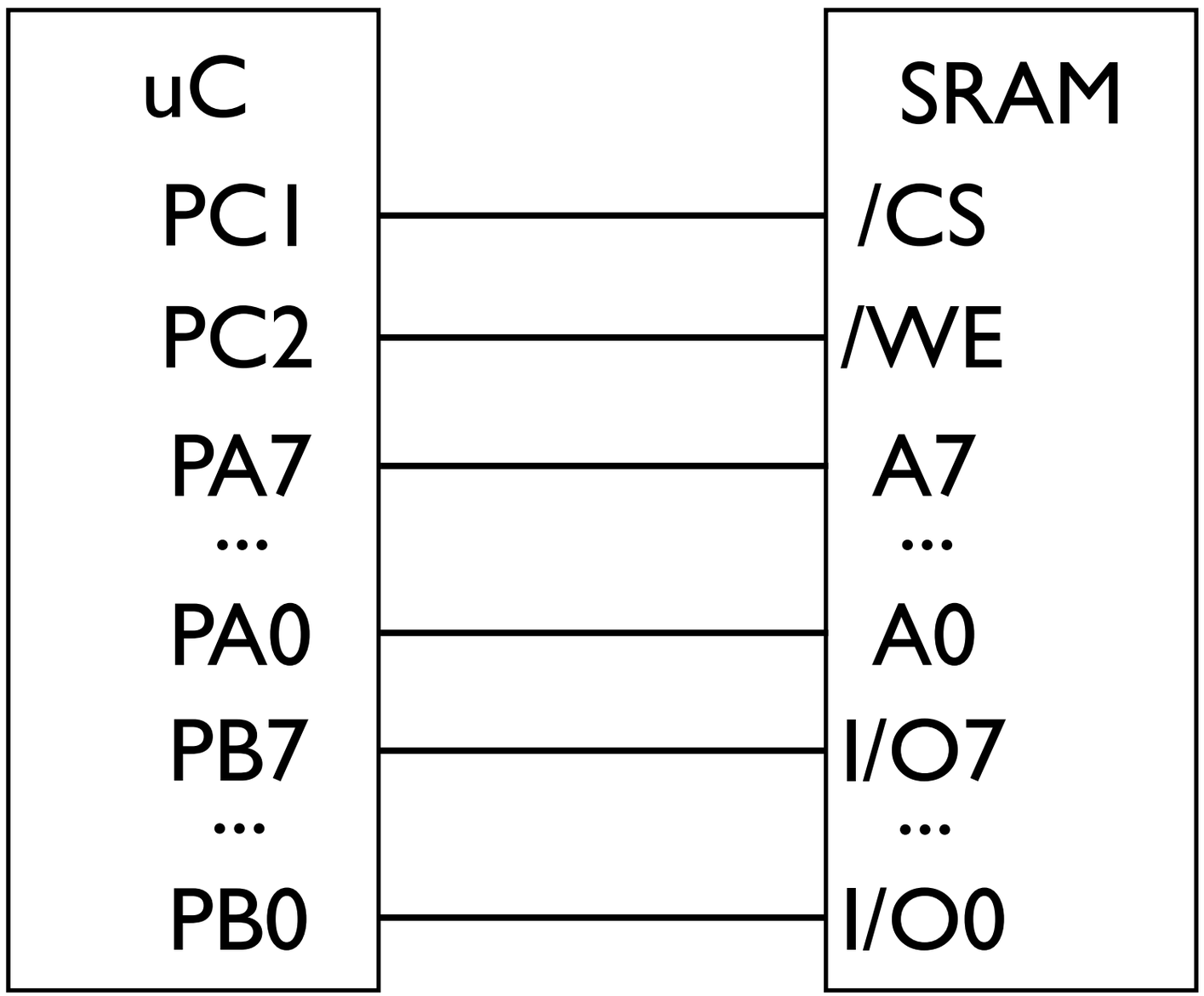
```
  A[7:0]=0x34
I/O[7:0]=0xAA
  CS=0, WE=0
```

example: SRAM

(old way)

write `0xAA` to address `0x34`:

```
PA[7:0]=0x34
PB[7:0]=0xAA
 PC1=0,PC2=0
```



our routine

ex: uC configured for mm with 256x8 device

addr: 8 bits
data: 8 bits

write `0xAA` to
`0xA0000034`

a write to mm location:

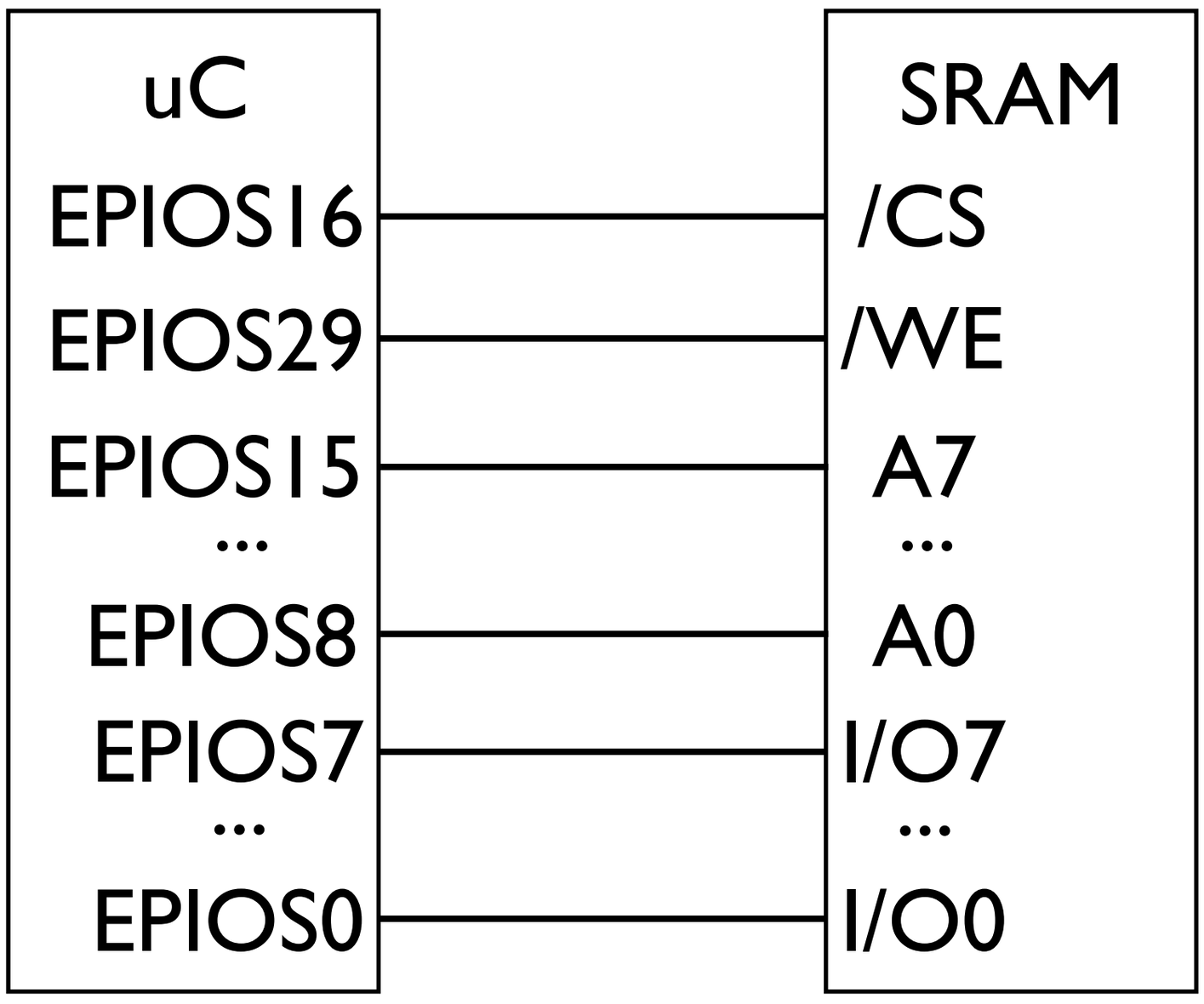
1. addr = `0xA0000034 & 0xFF = 0x34`
1. data = `0xAA & 0xFF = 0xAA`

what device will see
(bits uC sets on pins)

example: SRAM

(mm way)

our routine ⟶ write `0xAA` to address `0xA0000034`



uC does ↓

```
EPIOS[15:8]=0x34
 EPIOS[7:0]=0xAA
      EPIOS16=0,
       EPIOS29=0
```

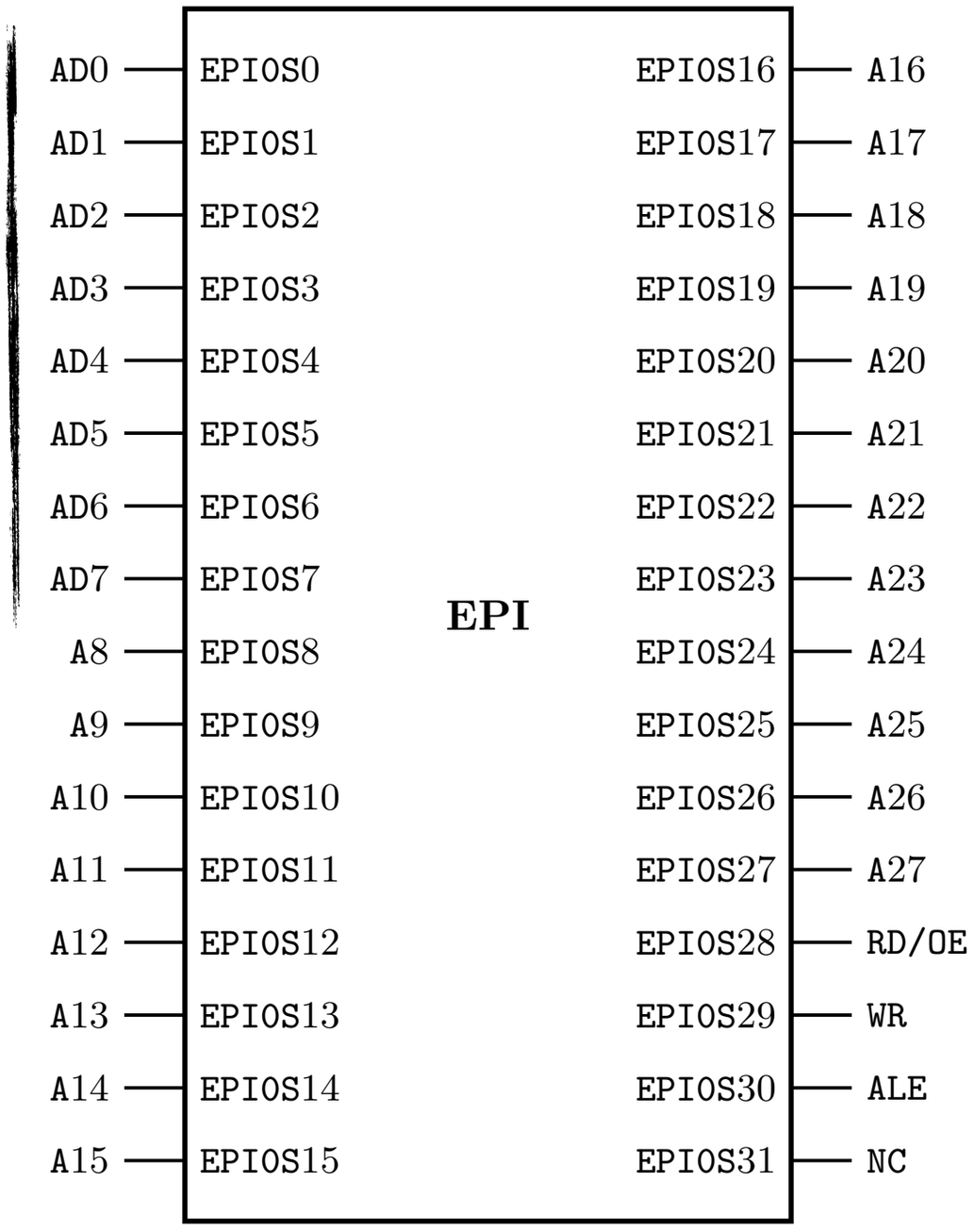
EPI: 8-bit multiplexed mode

8-bits

address and data

2^27 of something

(doesn't have to be bytes)



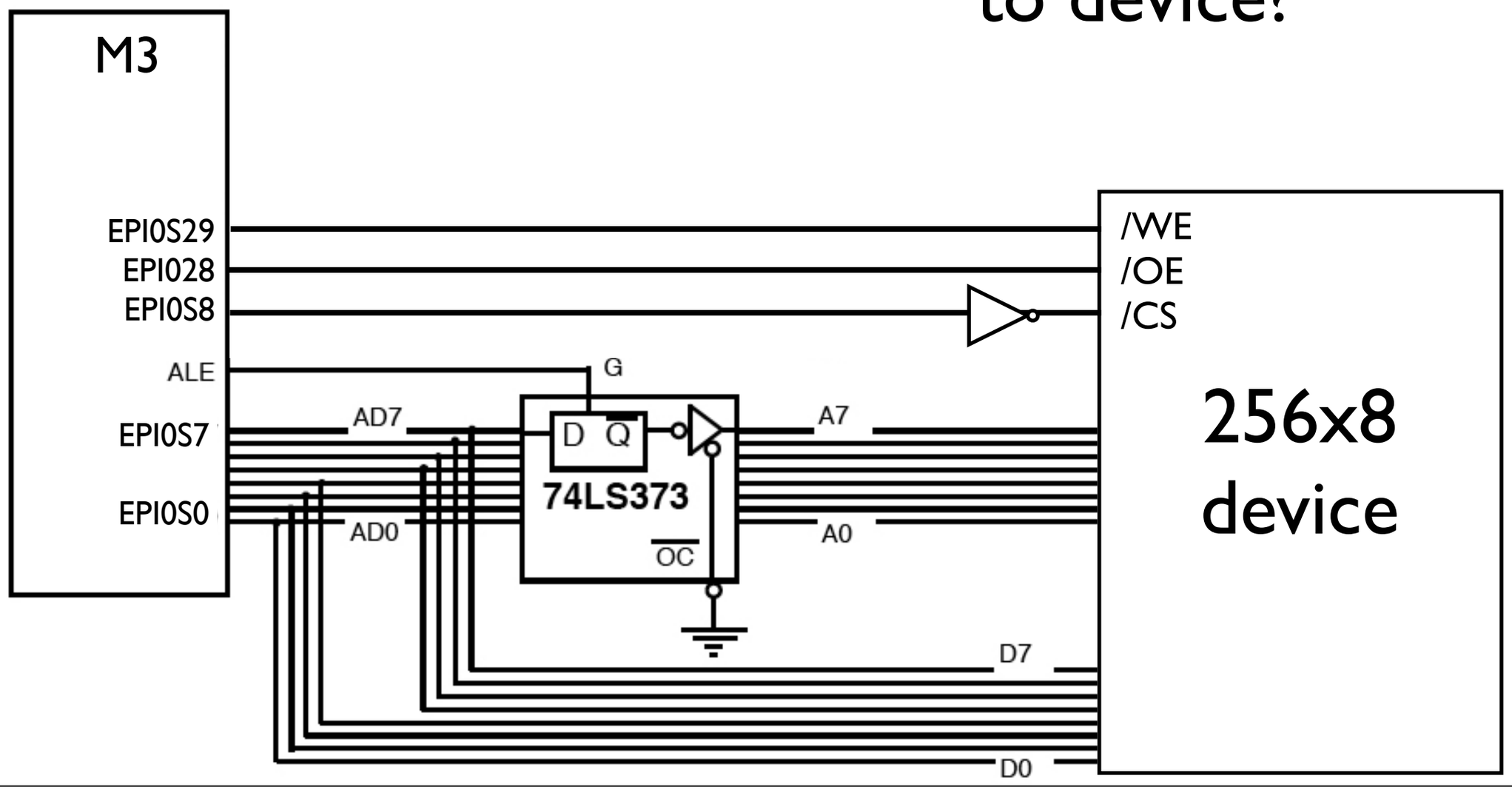
AD[7:0] = 0xA00000[00]/DATA[7:0]

A[27:8] = 0xA[00000]00

memory map
(multiplexed)

if base external mm addr: 0xA0000000

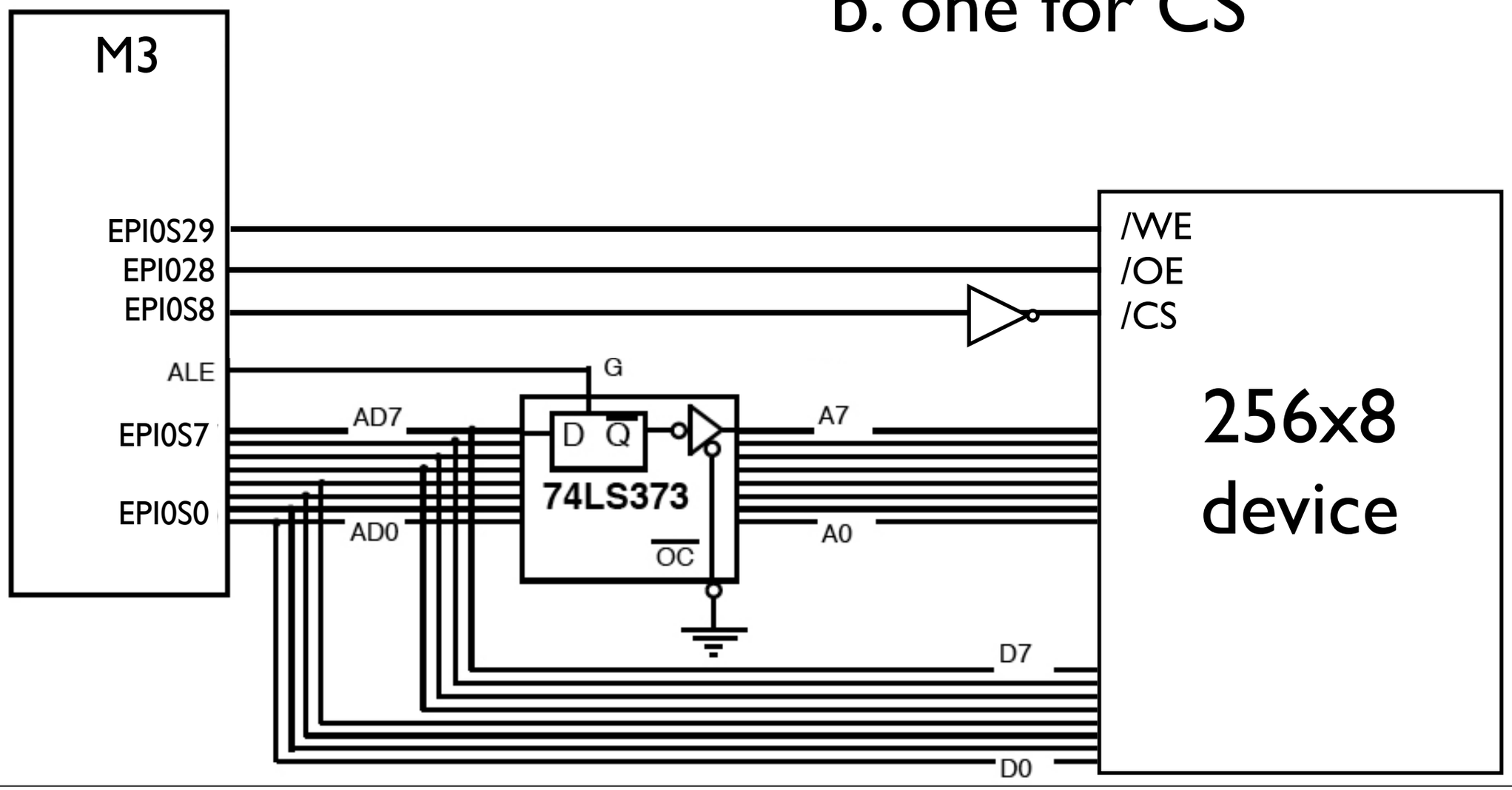
Q: which addr correspond to device?



memory map

Q: which addr correspond
to device?

to address device: 9 bits
a. eight for data addr
b. one for CS



memory map

Q: which addr correspond to device?

offsets for device:

0b100000000 -- 0b111111111

0x100 -- 0x1FF

for CS pin

device has addresses: 0xA0000100 -- 0xA00001FF

reading/writing to (MM) device in C

to make this work:

var. needs to be located/point to location in external memory

when data is assigned to var, uC puts it at the location or the location pointed at

ex: MM devices and C

(write `0xAA` to address `0x34`)

```
unsigned char X __attribute__((at(0xA0000134)));

void main()
{
   X = 0xAA;
}
```

```
ldr  R1,=0xA0000134
mov  R0,#0xAA
str R0,[R1]
```

ports/pins activated as before

stores `0xAA` in external memory at `0x34`

ex: simple way to access memory of device

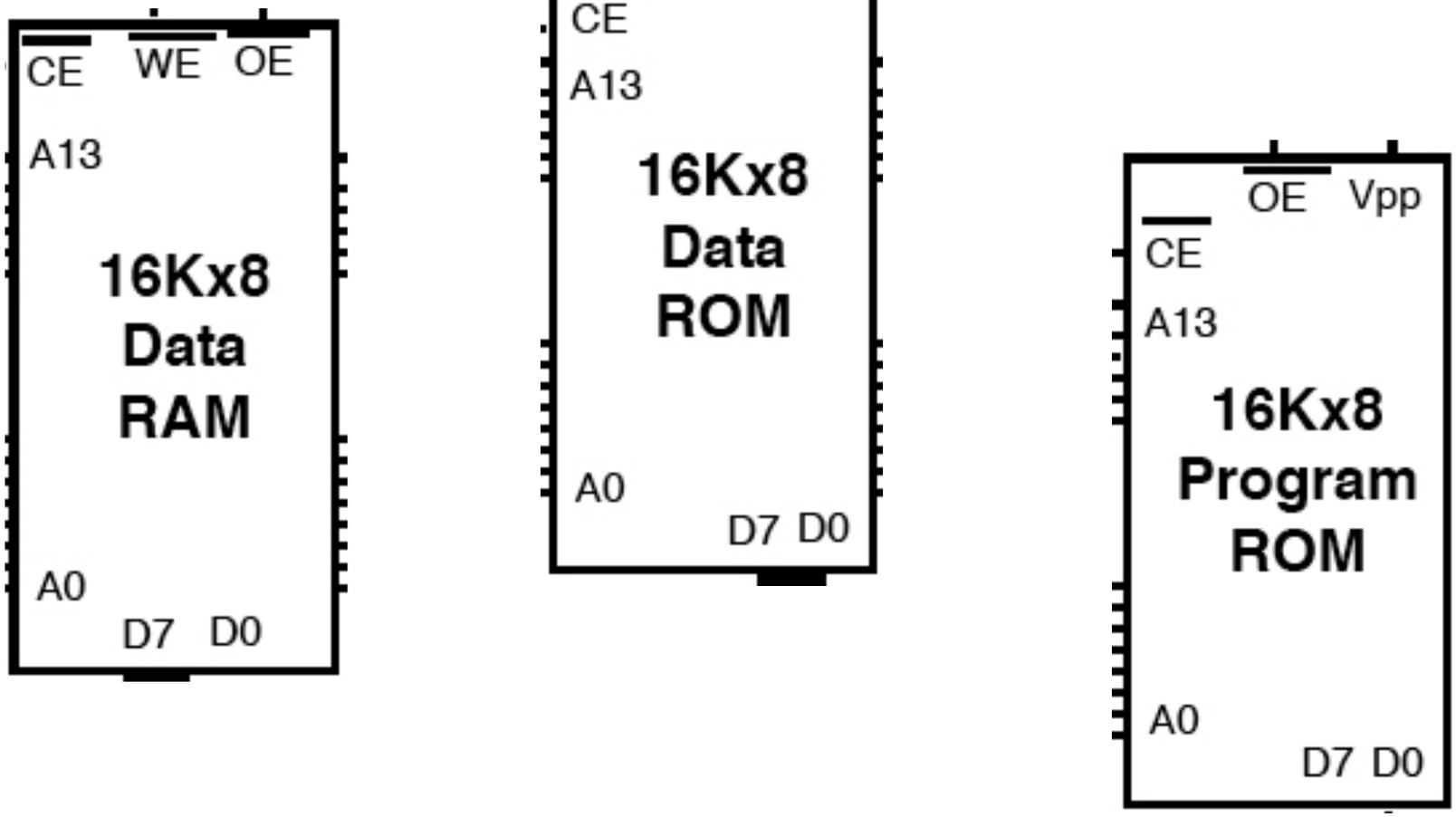
use pointer ⟶ then device appears as array

```
unsigned char *DEV = (unsigned char *) 0xA0000100;

void main()
{
  unsigned char i;

  /* write 0...255 in external memory at 0x0 to 0xFF */
  for(i = 0;i<255;i++)
    DEV[i]=i;
}
```

Q: what if multiple devices?

(shared bus)

#bits for address =14
16K=2^14

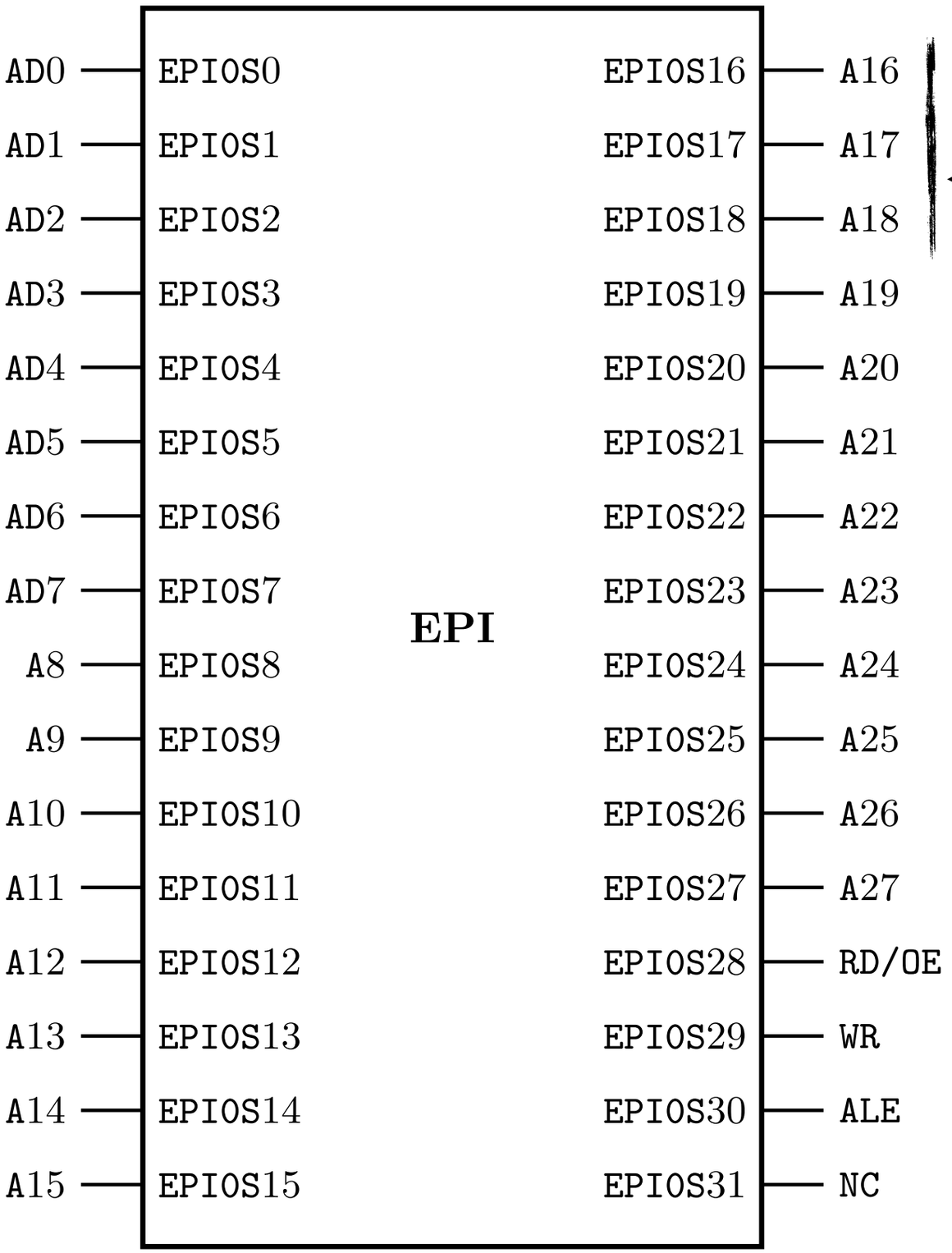


require 14 addr lines; have 28

a response:

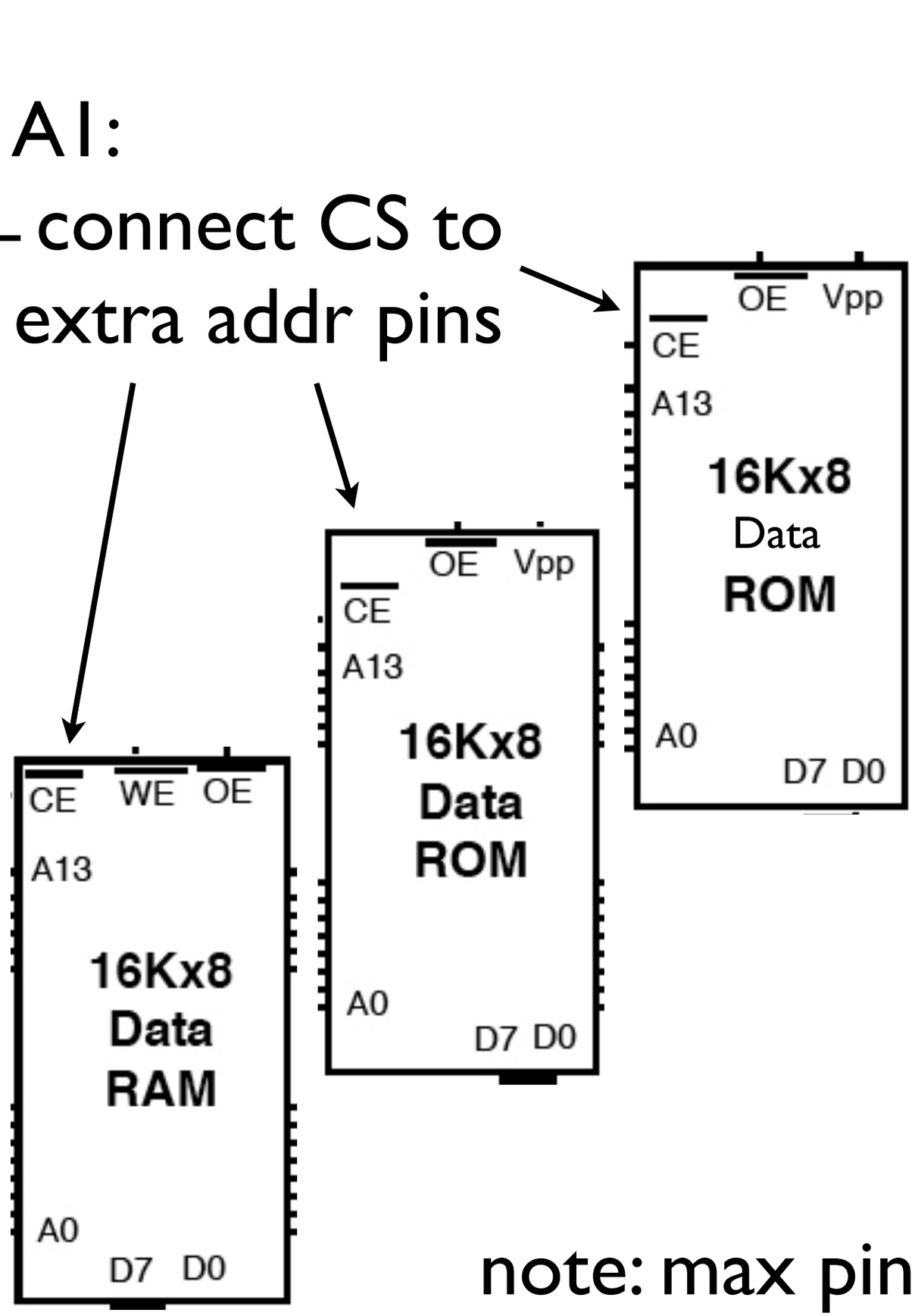
but not the right one

Q: what if multiple devices?



A1:

← connect CS to extra addr pins

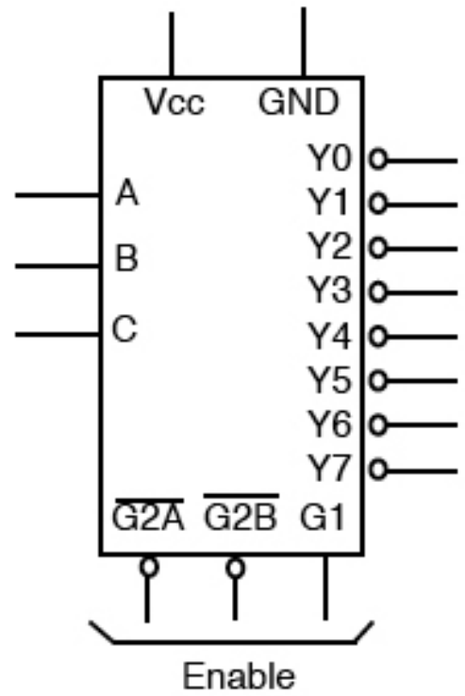


note: max pins

Q: what if multiple devices?

A2: decoder (minimise pins)

Block Diagram

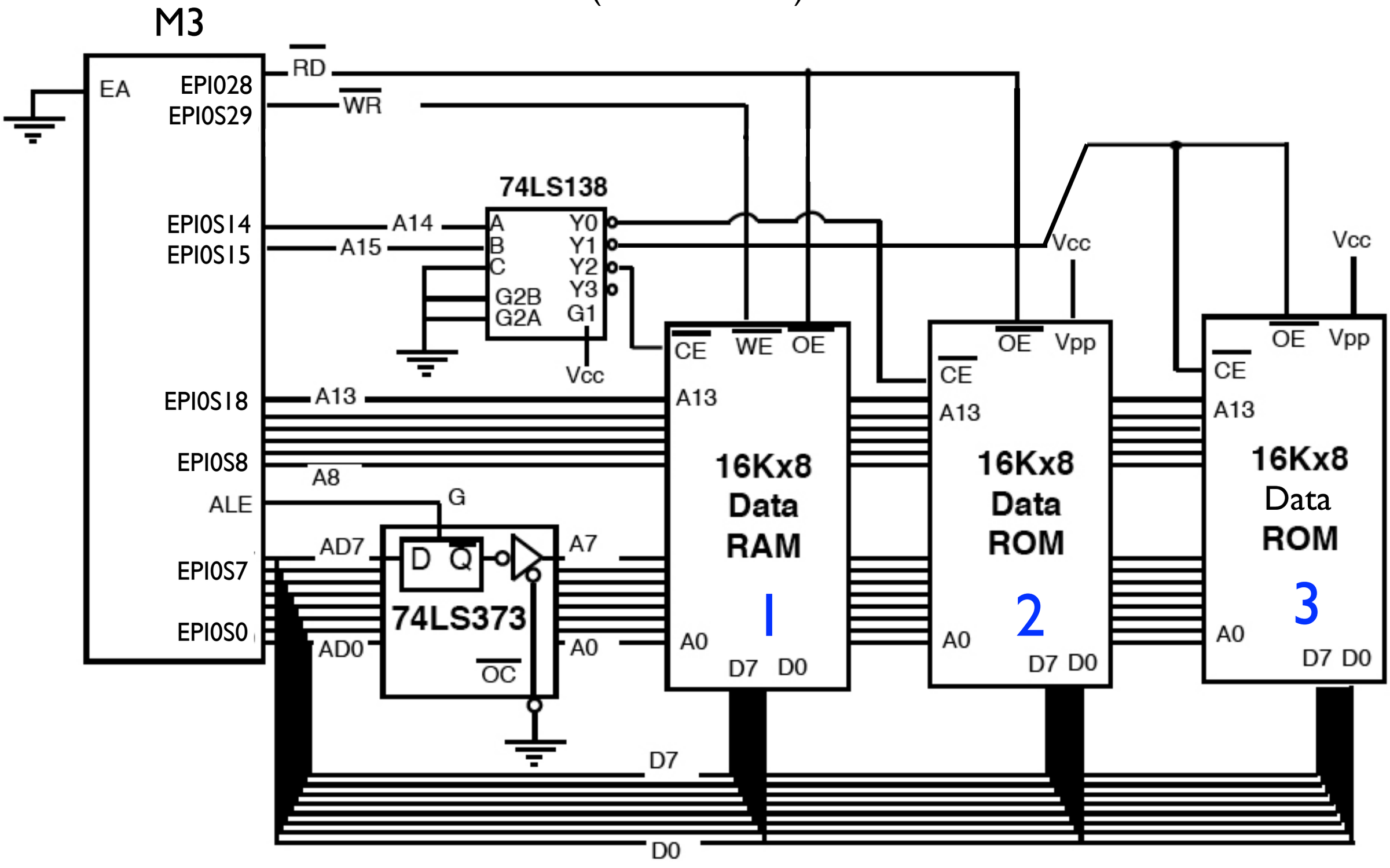


Function Table

Inputs			Outputs
Enable		Select	
G1	G2	C B A	Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7
X	H	X X X	H H H H H H H H
L	X	X X X	H H H H H H H H
H	L	L L L	**L** H H H H H H H
H	L	L L H	H **L** H H H H H H
H	L	L H L	H H **L** H H H H H
H	L	L H H	H H H **L** H H H H
H	L	H L L	H H H H **L** H H H
H	L	H L H	H H H H H **L** H H
H	L	H H L	H H H H H H **L** H
H	L	H H H	H H H H H H H **L**

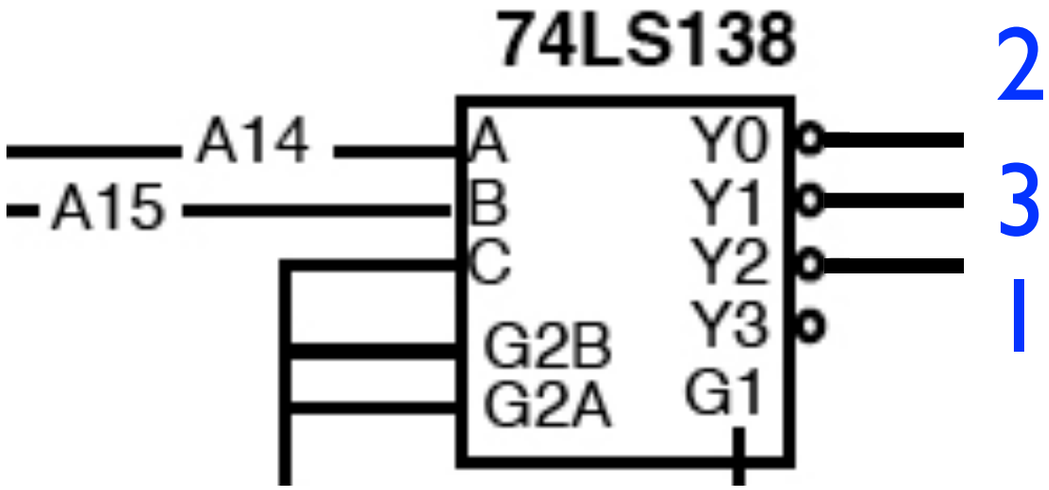
multiple devices

(2-bit decoder)



which addresses correspond to which device?

which bit combinations
activate devices?



when A[15:14]=[10] ⟶ device 1 on

when A[15:14]=[00] ⟶ device 2 on

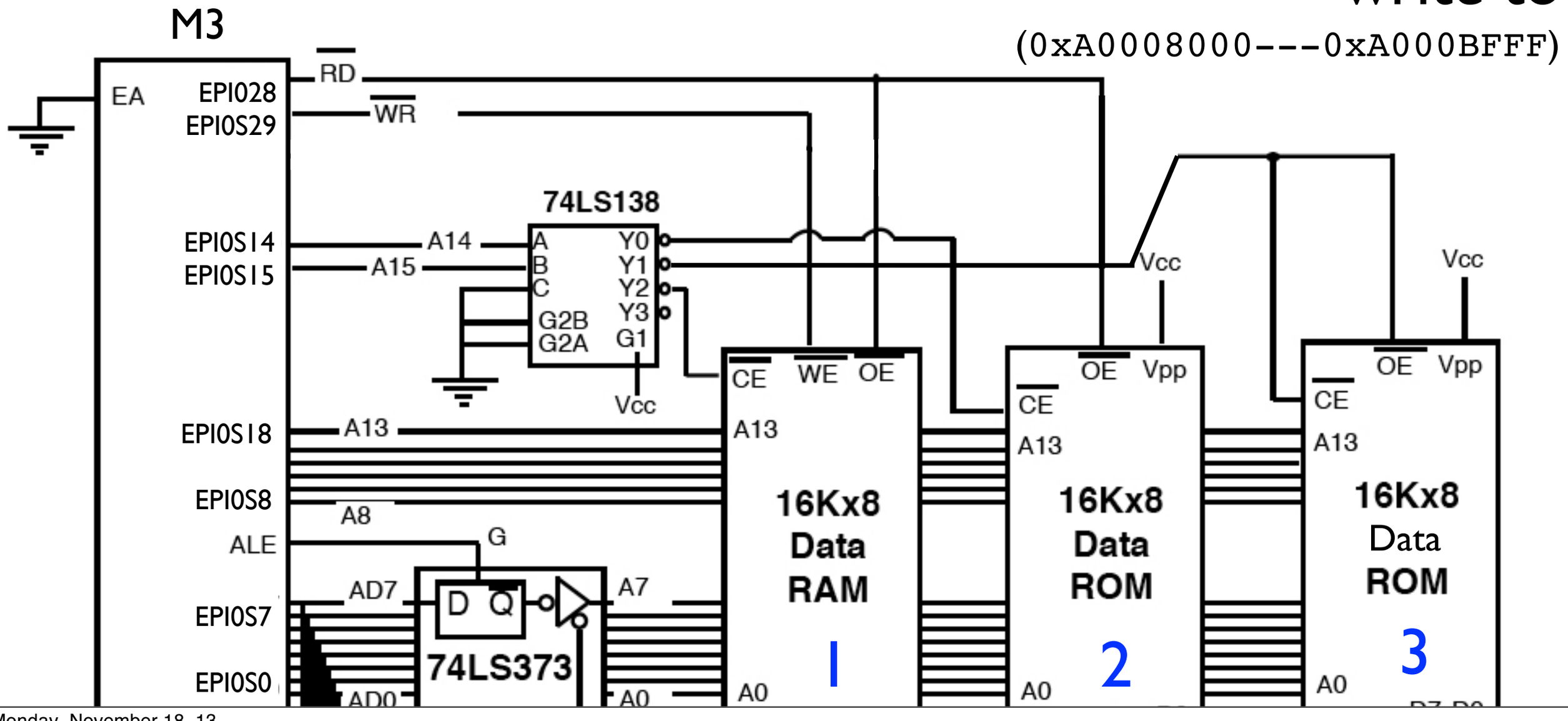
when A[15:14]=[01] ⟶ device 3 on

when A[15:14]=[10] ⟶ device 1 on

offsets
(from 0xA0000000)

when EPIOS[15:0]: 0b1000000000000000 to
0b1011111111111111

write to
(0xA0008000---0xA000BFFF)

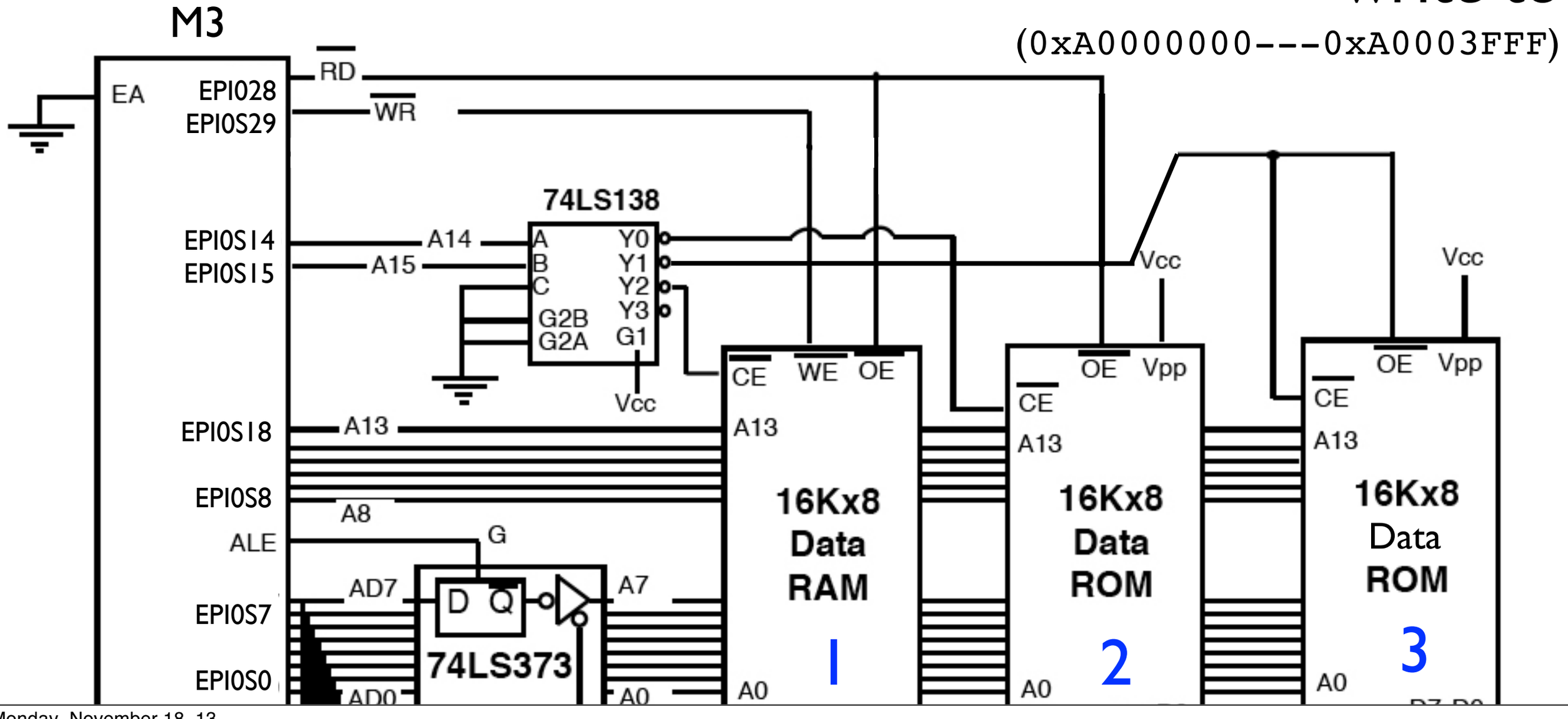


when A[15:14]=[00] ⟶ device 2 on

offsets
(from 0xA0000000)

when EPIOS[15:0]: 0b0000000000000000 to
0b0011111111111111

write to
(0xA0000000---0xA0003FFF)

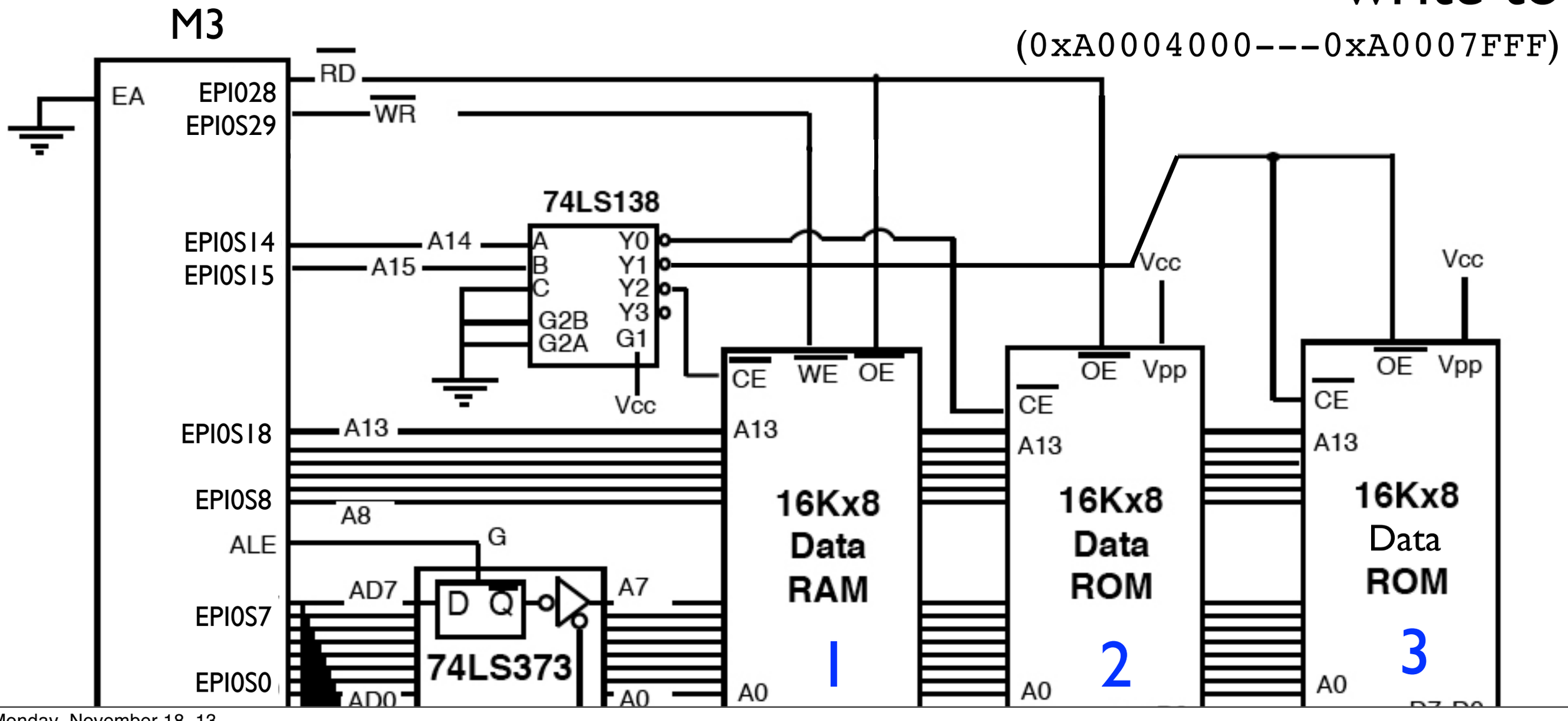


when A[15:14]=[01] ⟶ device 3 on

offsets
(from 0xA0000000)

when EPIOS[15:0]: 0b0100000000000000 to
0b0111111111111111

write to
(0xA0004000---0xA0007FFF)



ex: simple way to access memory of multiple devices

min addr for device 1:

0b1000000000000000=0x8000

pointers to devices

```
unsigned char *D1 = (unsigned char *) 0xA0008000;
unsigned char *D2 = (unsigned char *) 0xA0000000;
unsigned char *D3 = (unsigned char *) 0xA0004000;
```

min addr for device 3:

0b0100000000000000=0x4000

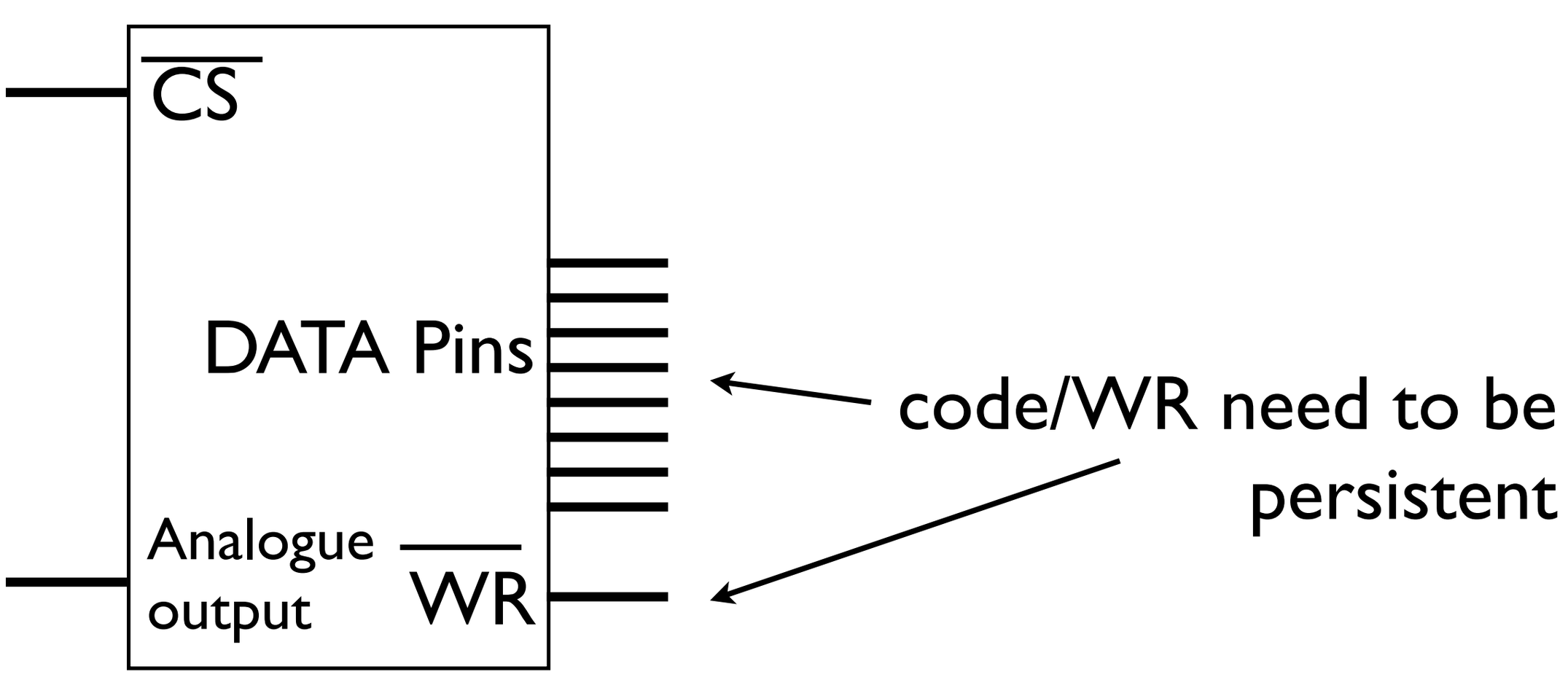
min addr for device 2:

0b0000000000000000=0x0000

access device:

Dx[0x0--0x3FFF]

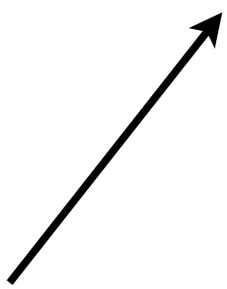
Q: how can we memory map a DAC?



Q: how can we memory map a DAC?

A:

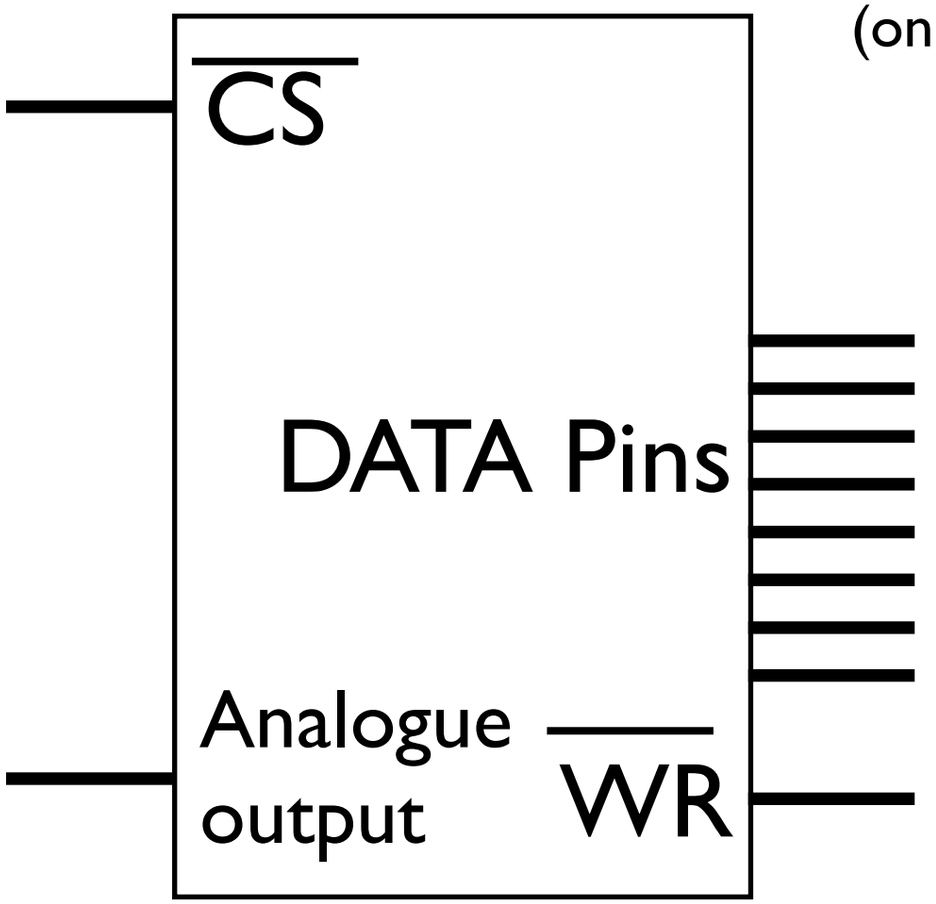
1. get DAC with memory
2. have something retain code/WR for us

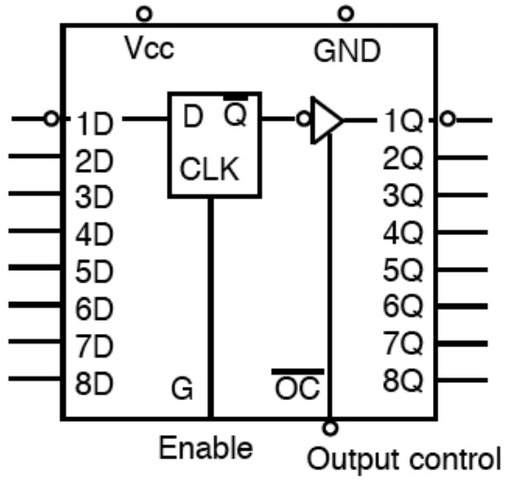


remember: uC only keeps data on pins temporarily

ex: mm a dac

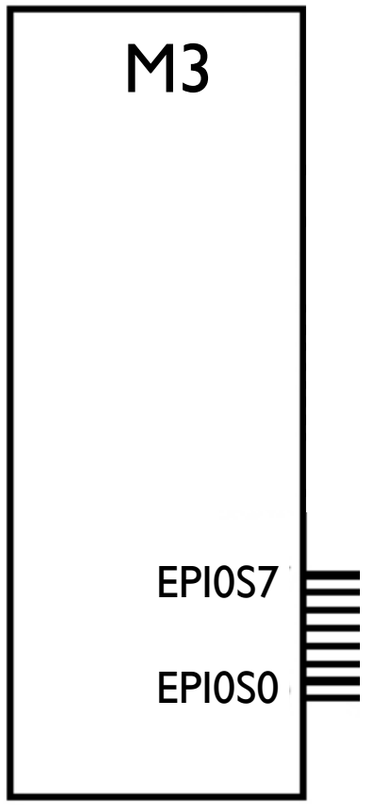
(on shared, multiplexed bus)





Funtion Table

Output control	Enable G	Enable D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z



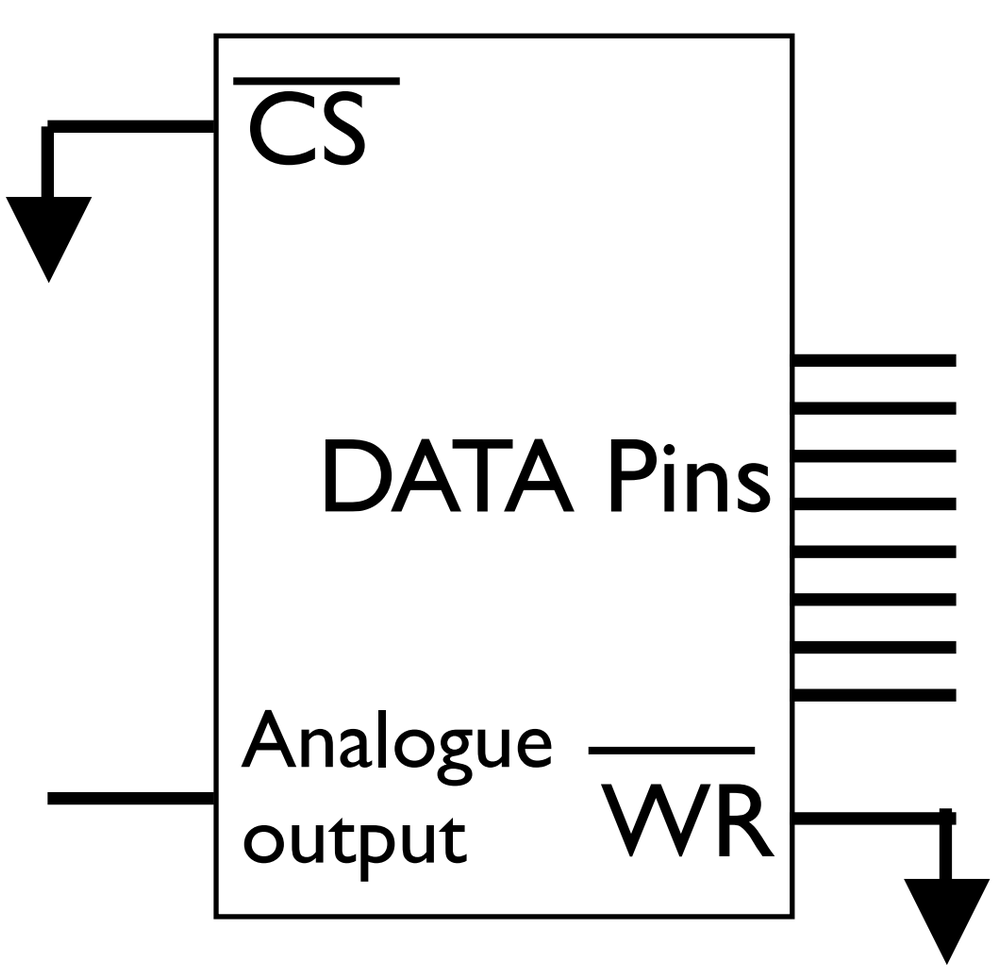
ex: mm a dac

(on shared, multiplexed bus)

dac:

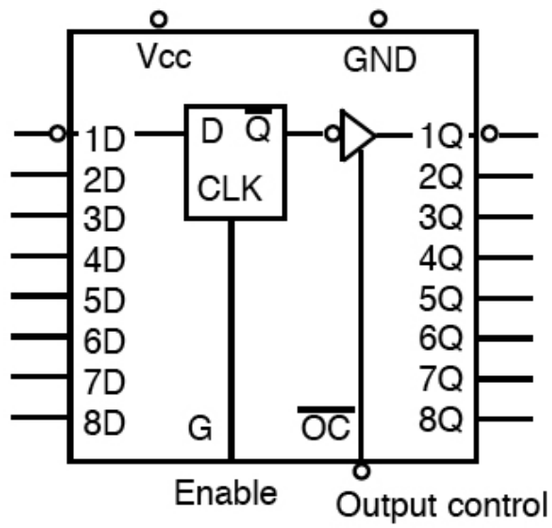
1. WR to ground
2. CS to ground

(always enabled, always outputting)



latch:

1. G(enable) to A8
2. xD (latch) to databus
3. xQ (latch) to dac input

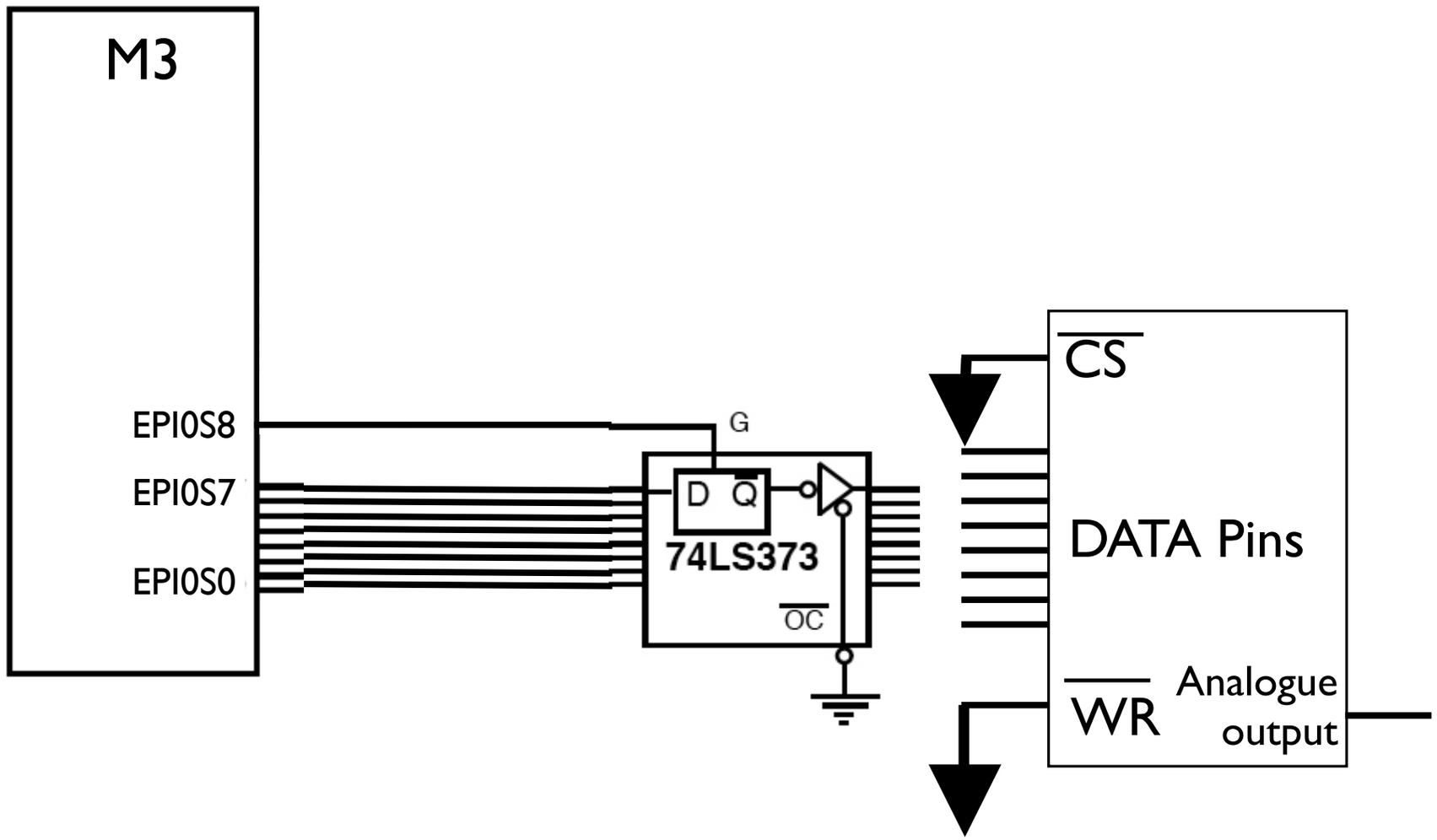


Funtion Table

Output control	Enable G	Enable D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

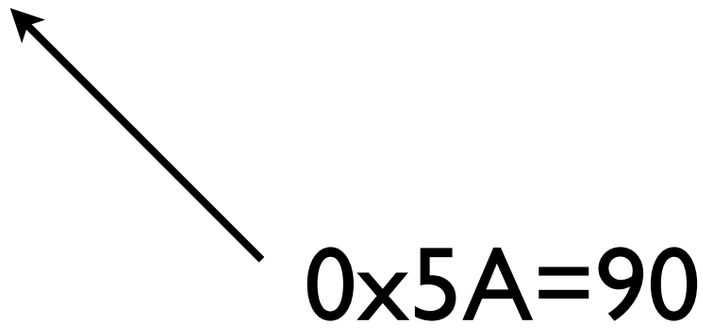
ex: mm a dac

(on shared, multiplexed bus)

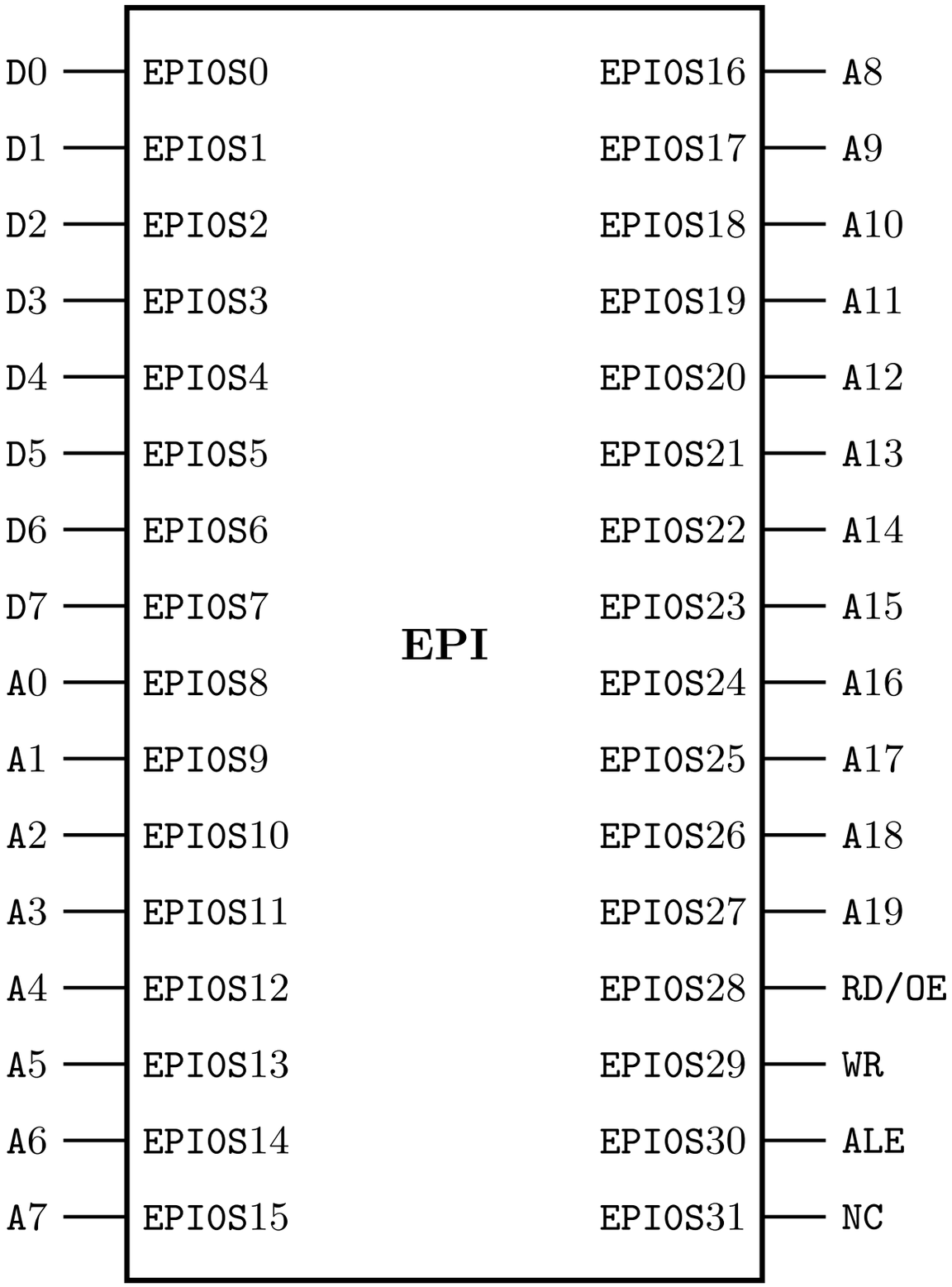


mm EPI0S[8:0]: `0b100000000` to `0b111111111`

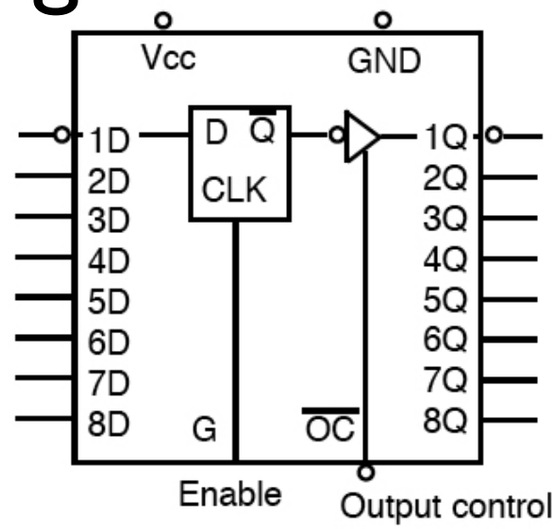
```
unsigned char DAC __attribute__((at(0xA0000100)));

DAC = 'Z';
```



can we do it without multiplexing?

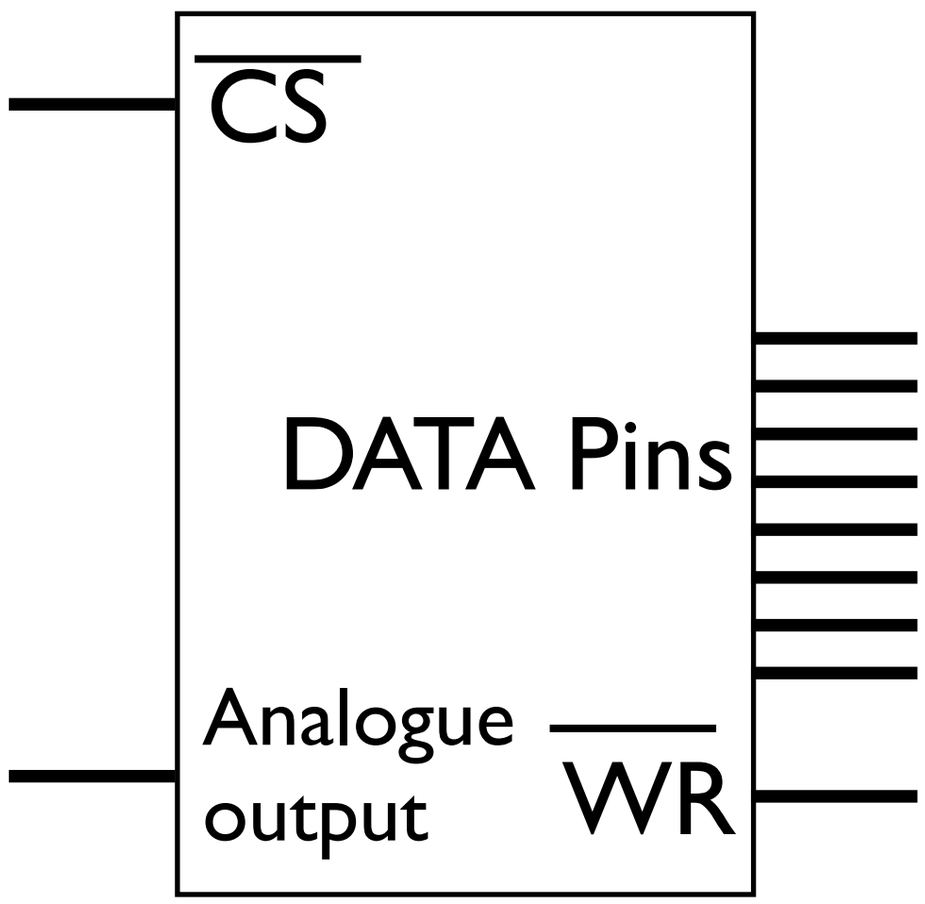


addr:0x10

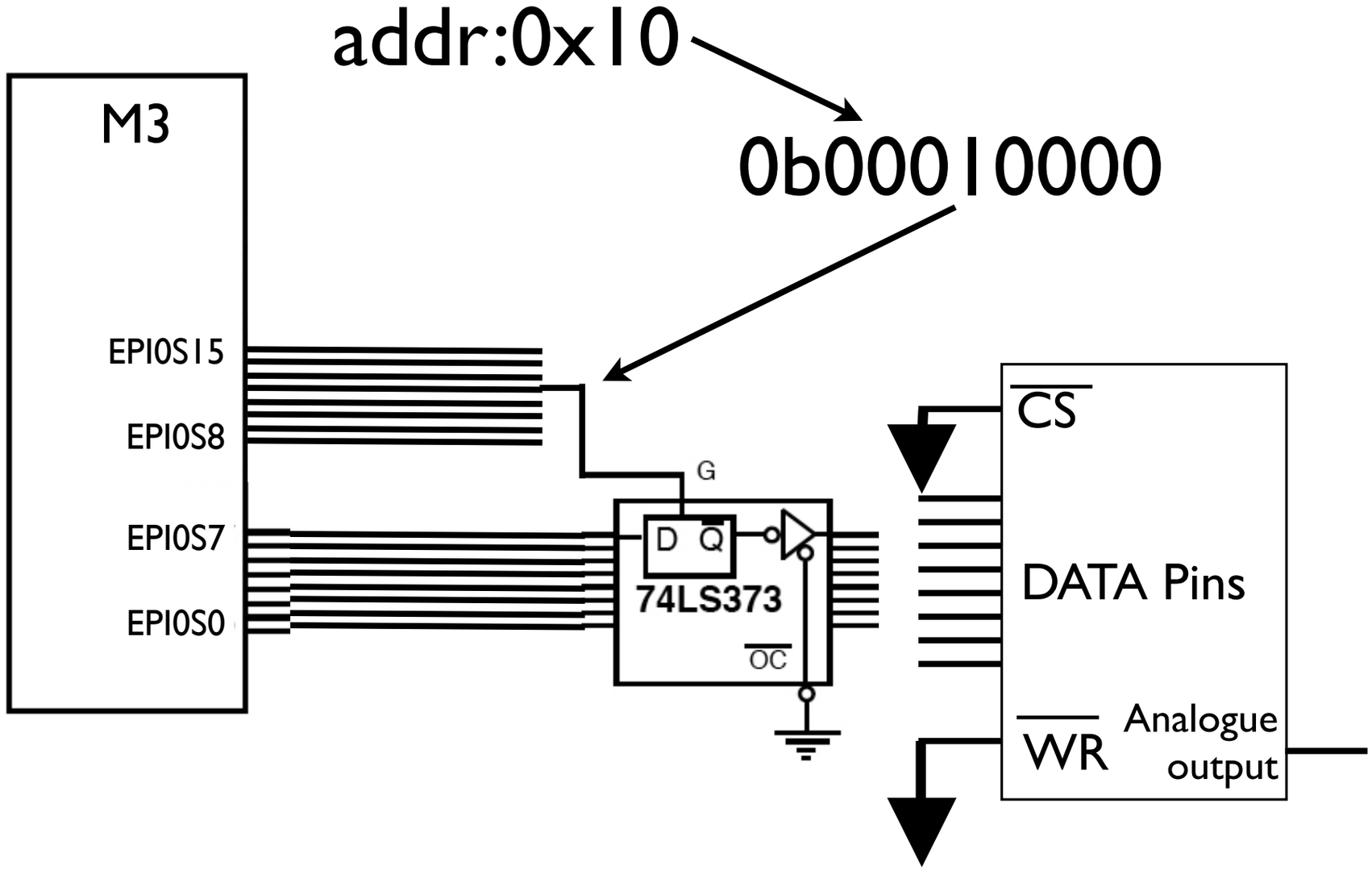


Funtion Table

Output control	Enable G	Enable D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z



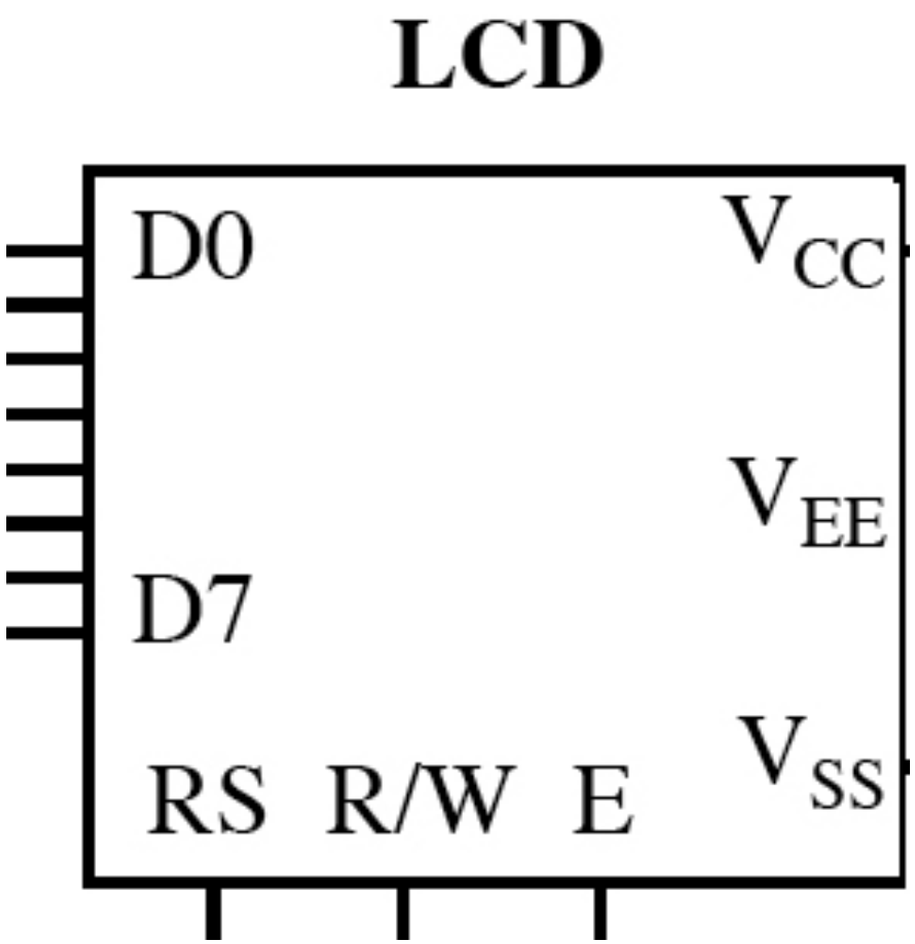
DAC without multiplexing




```
unsigned char DAC __attribute__((at(0xA0000010)));

DAC = 'Z';
```

mm of LCD

(demultiplexed)



command or chars

D0:D7 => 'data'

RS => D0:D7 is

0 command

1 data

RW => from LCD

0 write

1 read

E => CS/CE

these must be set for each communication