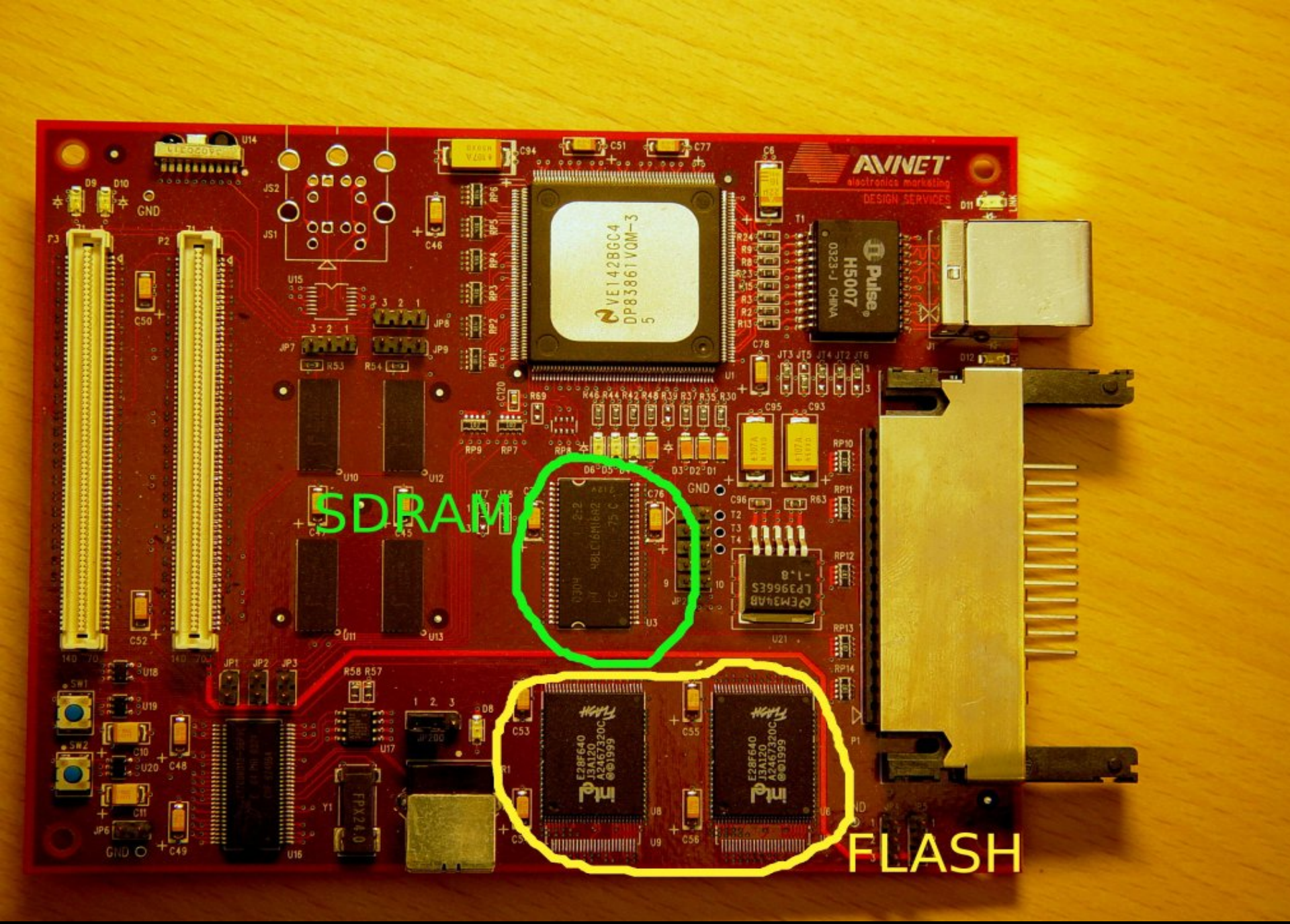
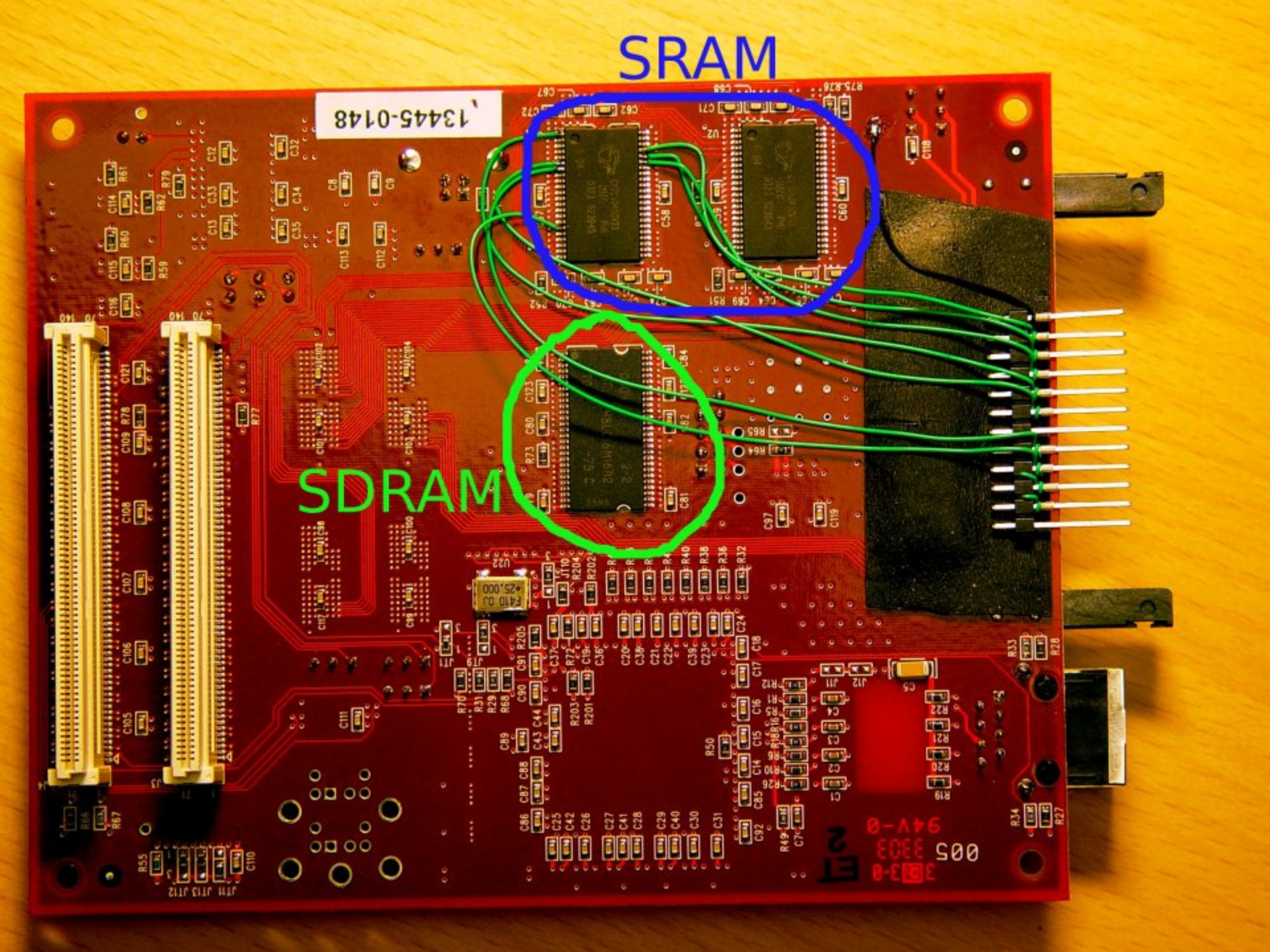
Lecture 8

- A memory controller
- Lab4 – a special instruction

PKMC


Wishbone bus
Memory bus
adr
dat_o
dat_i
stb
ack
...
Per Karlströms
Memory Controller
adr
dat_io
cs_sdram
cs_sram
cs_flash
...

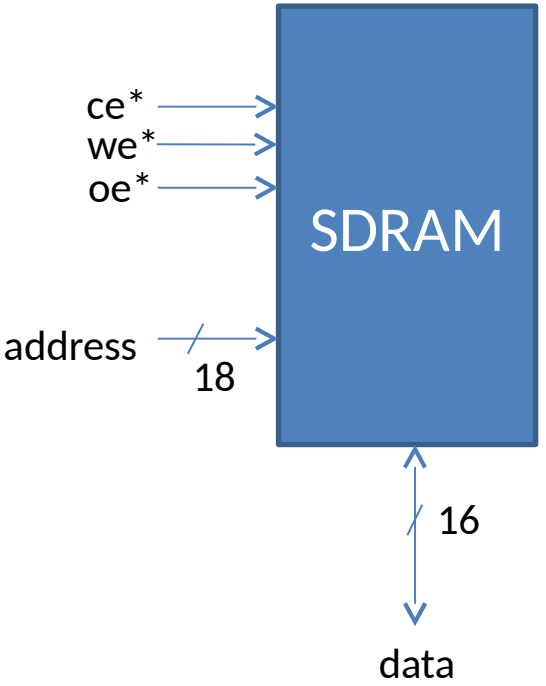

SDRAM
FLASH
AVNET
electronics marketing
DESIGN SERVICES
Pulse
H5007
0323-J CHINA
VE142BGC4
DP83861VQM-3
5
GND
U14
U15
U10
U12
U11
U13
U3
U1
U21
U16
U17
U18
U19
U20
U8
U9
U6
JS1
JS2
JP1
JP2
JP3
JP6
JP7
JP8
JP9
P1
P2
P3
SW1
SW2
Y1
FPX24.0
E28F640
J3A120
A24673320C
©©1999
Intel
FLASH


SRAM
SDRAM

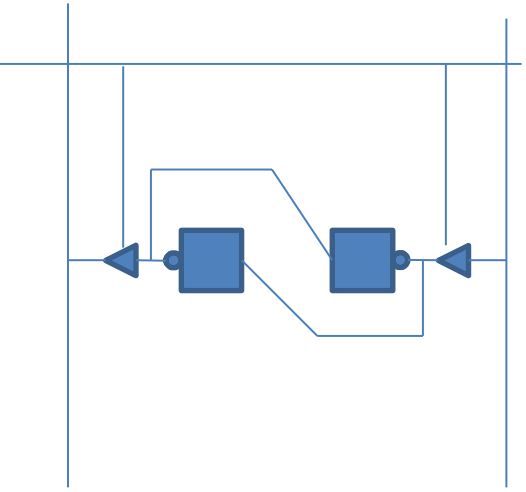
SRAM

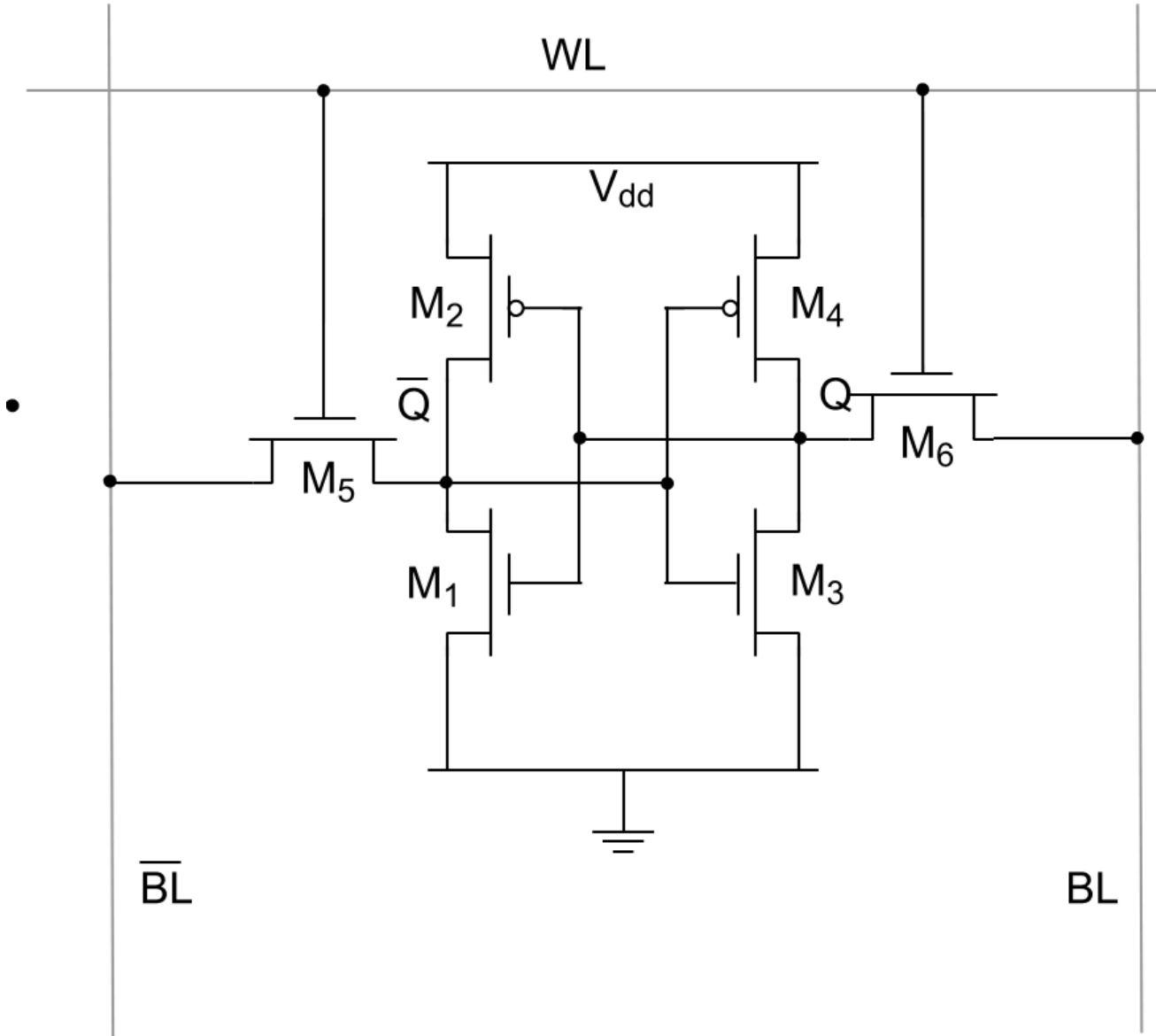
Static RAM

- Asynchronous device
- Memory element: Latch
- 2 x (256k x 16) = 1MB

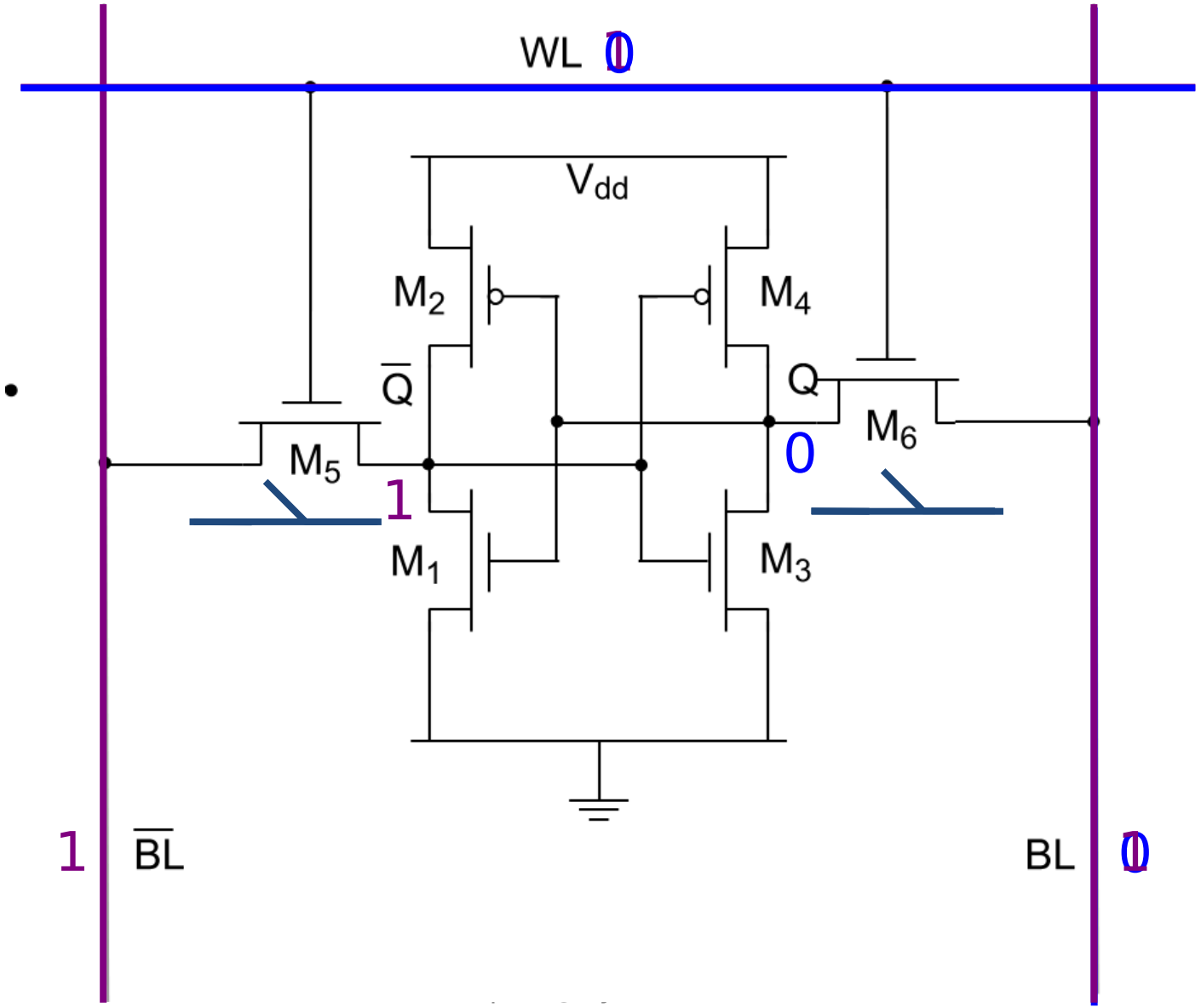


SRAM - Cell

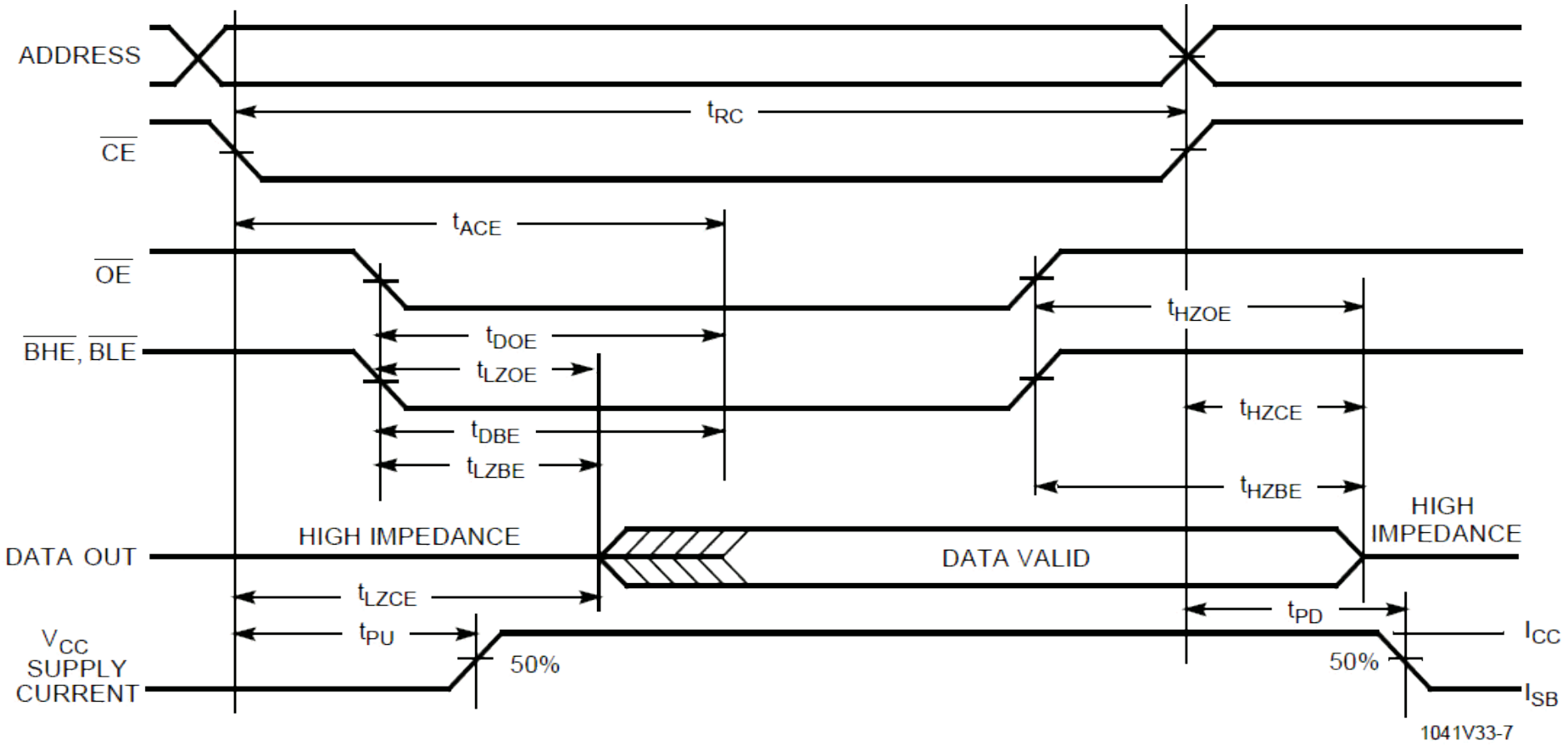




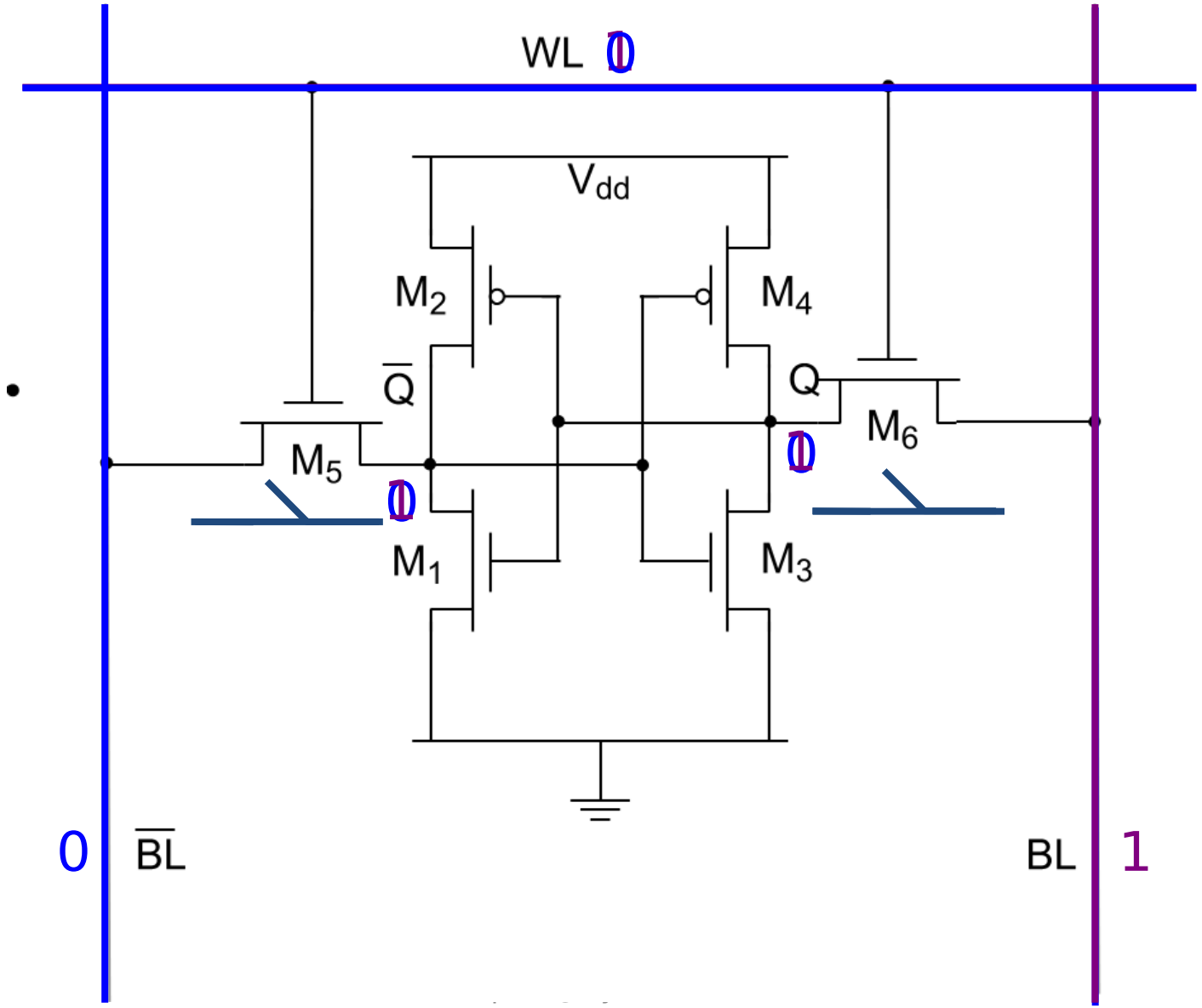
SRAM - Read



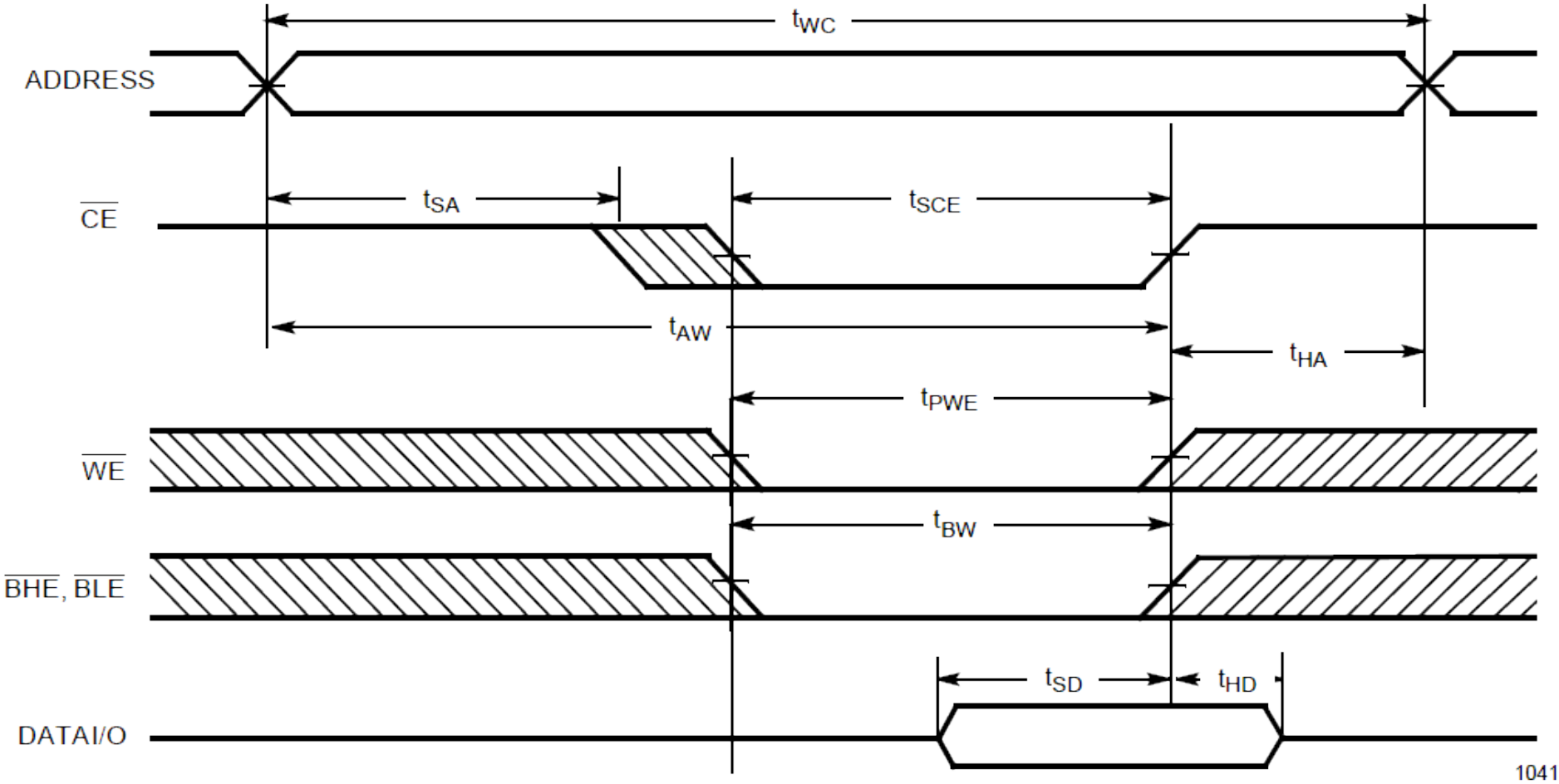
SRAM - Read


ADDRESS
t_{RC}
$\overline{CE}$
t_{ACE}
$\overline{OE}$
t_{HZOE}
t_{DOE}
$\overline{BHE}$, $\overline{BLE}$
t_{LZOE}
t_{HZCE}
t_{DBE}
t_{LZBE}
t_{HZBE}
HIGH IMPEDANCE
HIGH IMPEDANCE
DATA OUT
DATA VALID
t_{LZCE}
t_{PD}
t_{PU}
I_{CC}
V_{CC} SUPPLY CURRENT
50%
50%
I_{SB}
1041V33-7

SRAM - Write



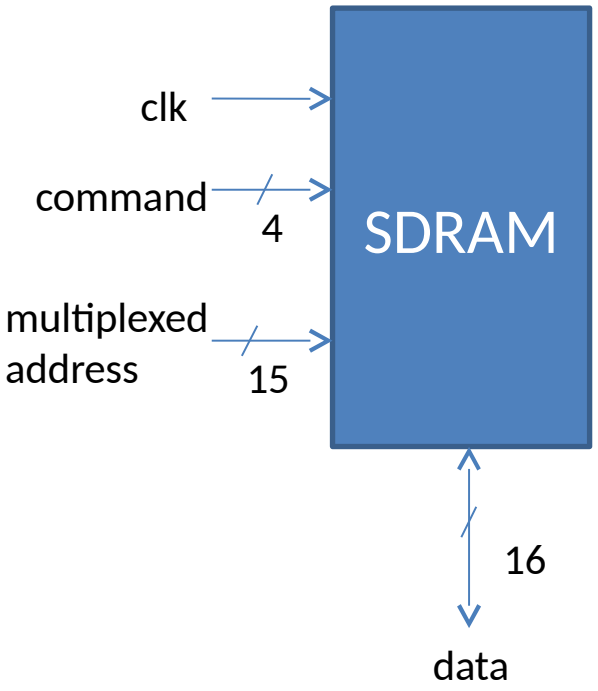
SRAM - Write


t_{WC}
ADDRESS
$\overline{CE}$
t_{SA}
t_{SCE}
t_{AW}
t_{HA}
t_{PWE}
$\overline{WE}$
t_{BW}
$\overline{BHE}$, $\overline{BLE}$
t_{SD}
t_{HD}
DATAI/O
1041

SDRAM

Synchronuous Dynamic RAM

- Clocked device
- Memory element: Capacitance
- Needs periodic refreshing
- Pipelined operation
- Burst oriented
  - Single burst in our design
- 2 x (16M x 16) = 64MB



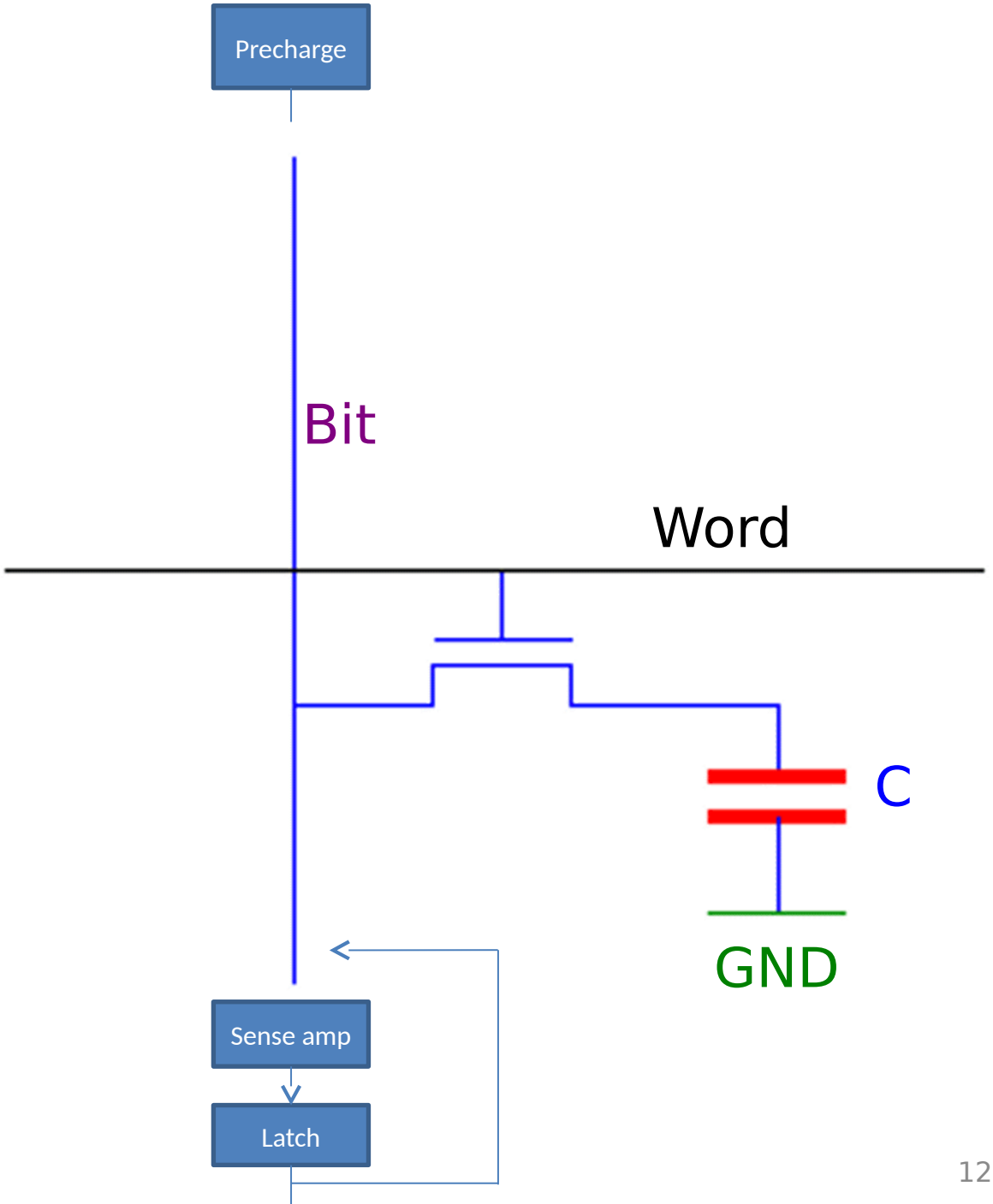
SDRAM

Sketchy read cycle

1. Precharge bit line to V/2
2. Let bit line float
3. Connect sense amp to bit line
4. Connect C to bit line
5. Hold value in latch
6. Write value back to C

Refresh cycle

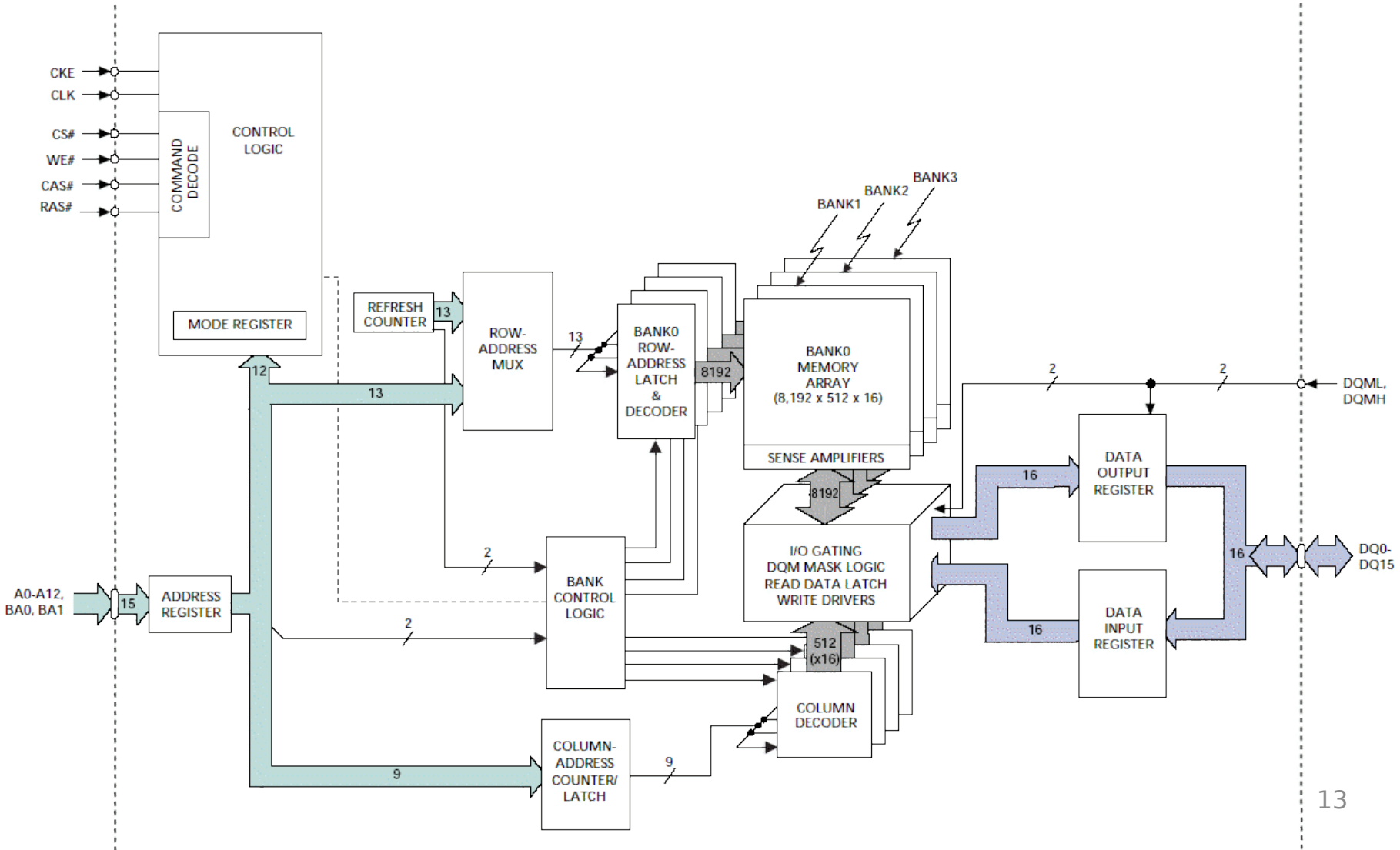
Dummy read cycle



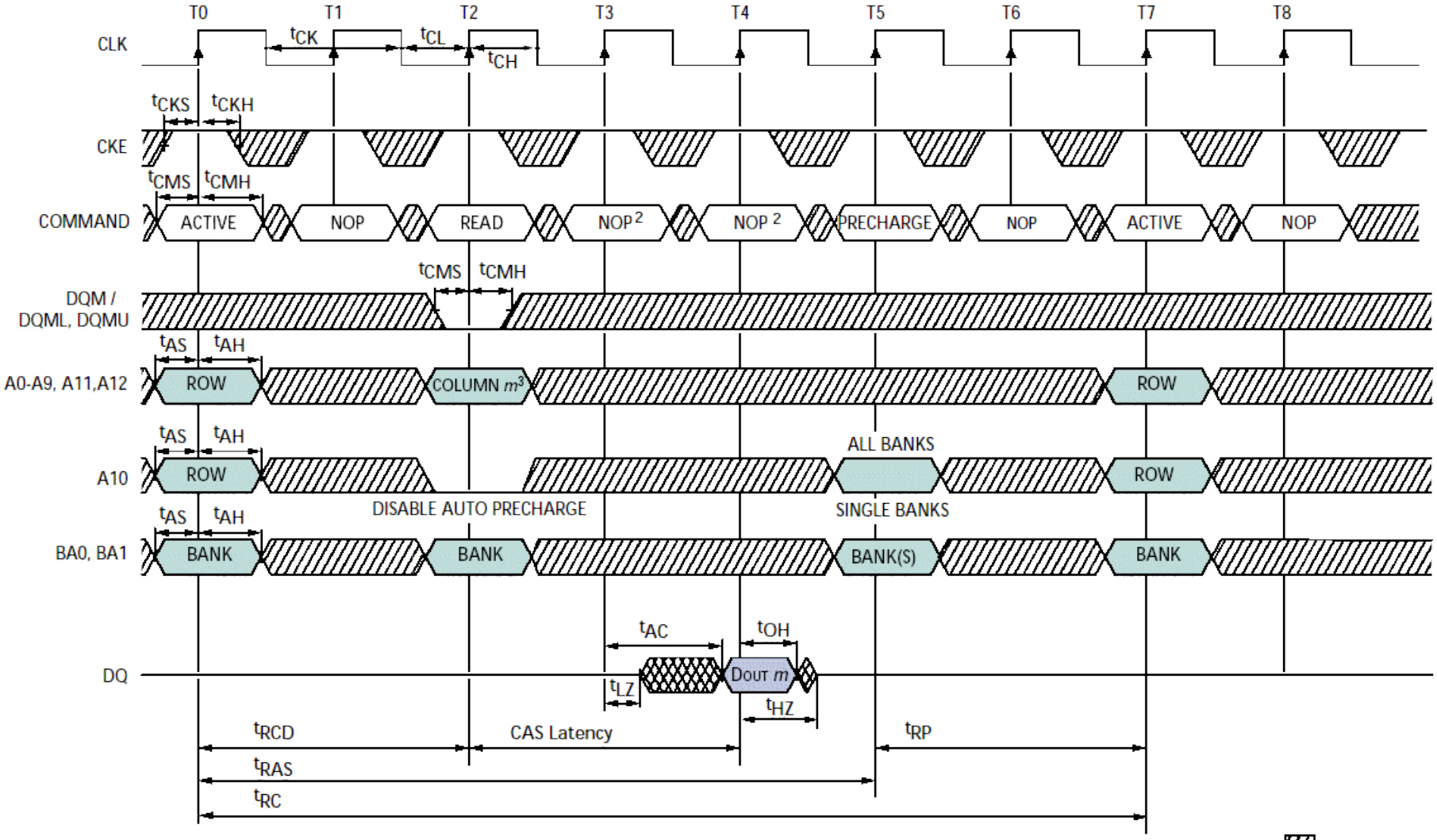
SDRAM Architecture

32 MB = 16 MW per chip, address bits = 24 = 13+11

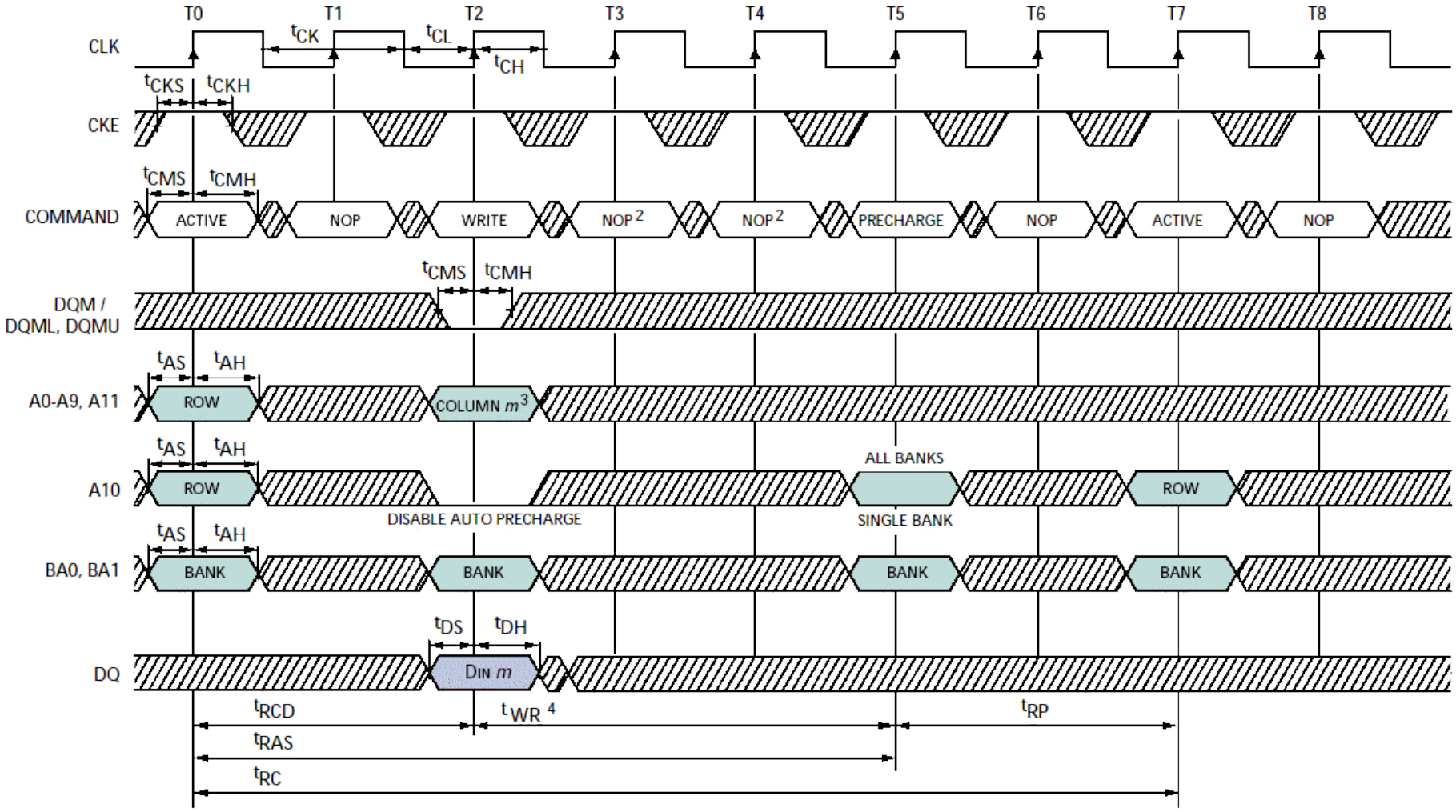
burst oriented



SDRAM Read

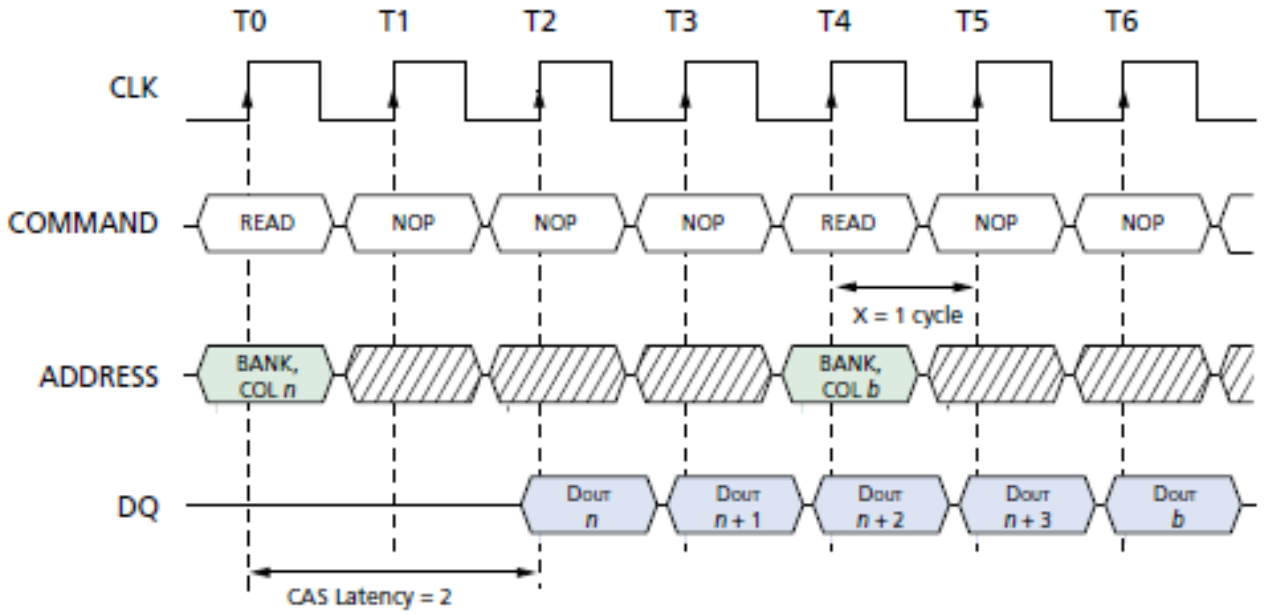

T0
T1
T2
T3
T4
T5
T6
T7
T8
CLK
tCK
tCL
tCH
tCKS
tCKH
CKE
tCMS
tCMH
COMMAND
ACTIVE
NOP
READ
NOP 2
NOP 2
PRECHARGE
NOP
ACTIVE
NOP
tCMS
tCMH
DQM / DQML, DQMU
tAS
tAH
A0-A9, A11,A12
ROW
COLUMN m3
ROW
tAS
tAH
A10
ROW
ALL BANKS
ROW
DISABLE AUTO PRECHARGE
SINGLE BANKS
tAS
tAH
BA0, BA1
BANK
BANK
BANK(S)
BANK
tAC
tOH
DQ
tLZ
DOUT m
tHZ
tRCD
CAS Latency
tRP
tRAS
tRC
DON'T CARE
UNDEFINED

SDRAM Write


T0
T1
T2
T3
T4
T5
T6
T7
T8
CLK
tCK
tCL
tCH
tCKS
tCKH
CKE
tCMS
tCMH
COMMAND
ACTIVE
NOP
WRITE
NOP 2
NOP 2
PRECHARGE
NOP
ACTIVE
NOP
tCMS
tCMH
DQM / DQML, DQMU
tAS
tAH
A0-A9, A11
ROW
COLUMN m 3
tAS
tAH
A10
ROW
ALL BANKS
ROW
DISABLE AUTO PRECHARGE
SINGLE BANK
tAS
tAH
BA0, BA1
BANK
BANK
BANK
BANK
tDS
tDH
DQ
DIN m
tRCD
tWR 4
tRP
tRAS
tRC

Consecutive read bursts

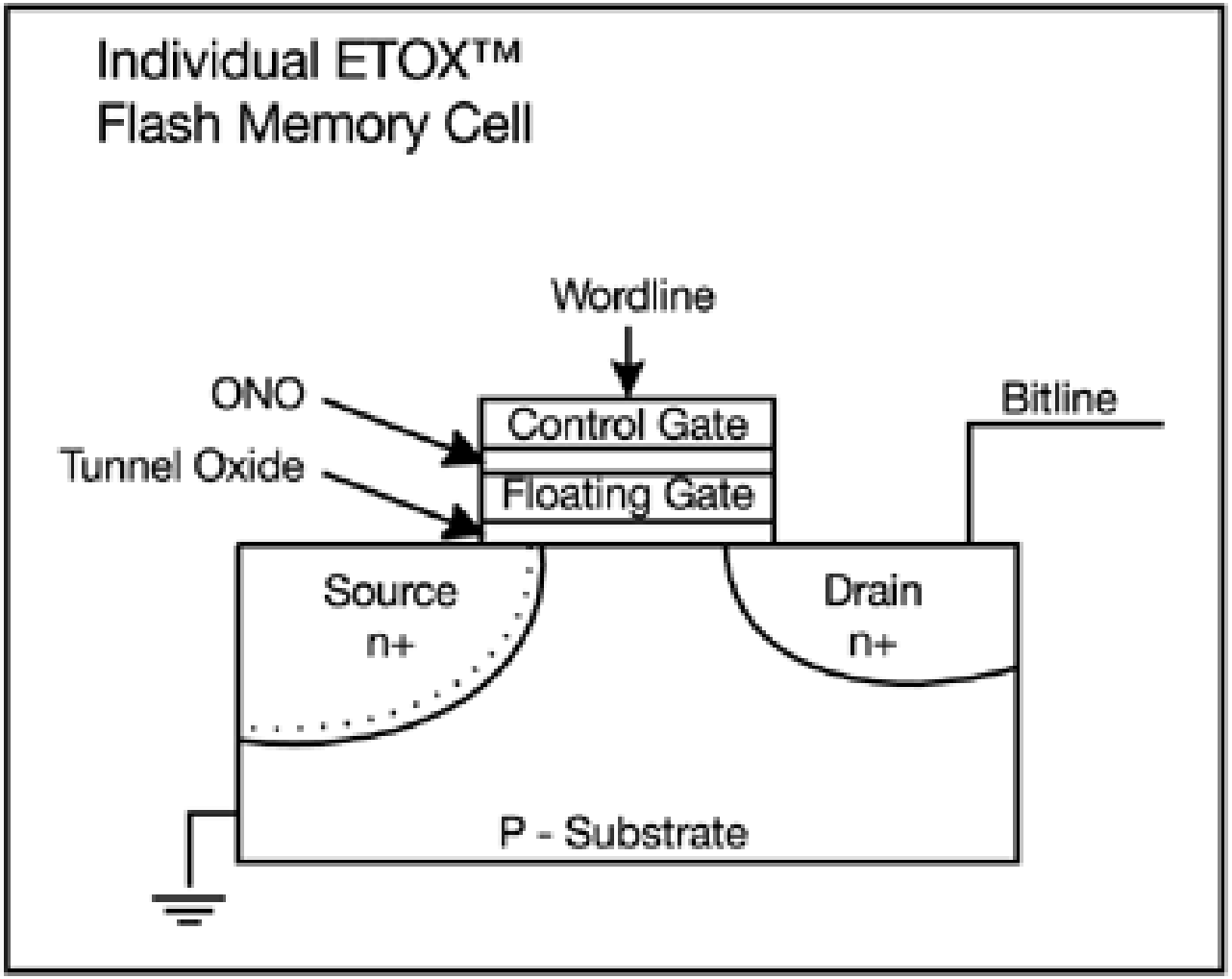
Mode register must be programmed



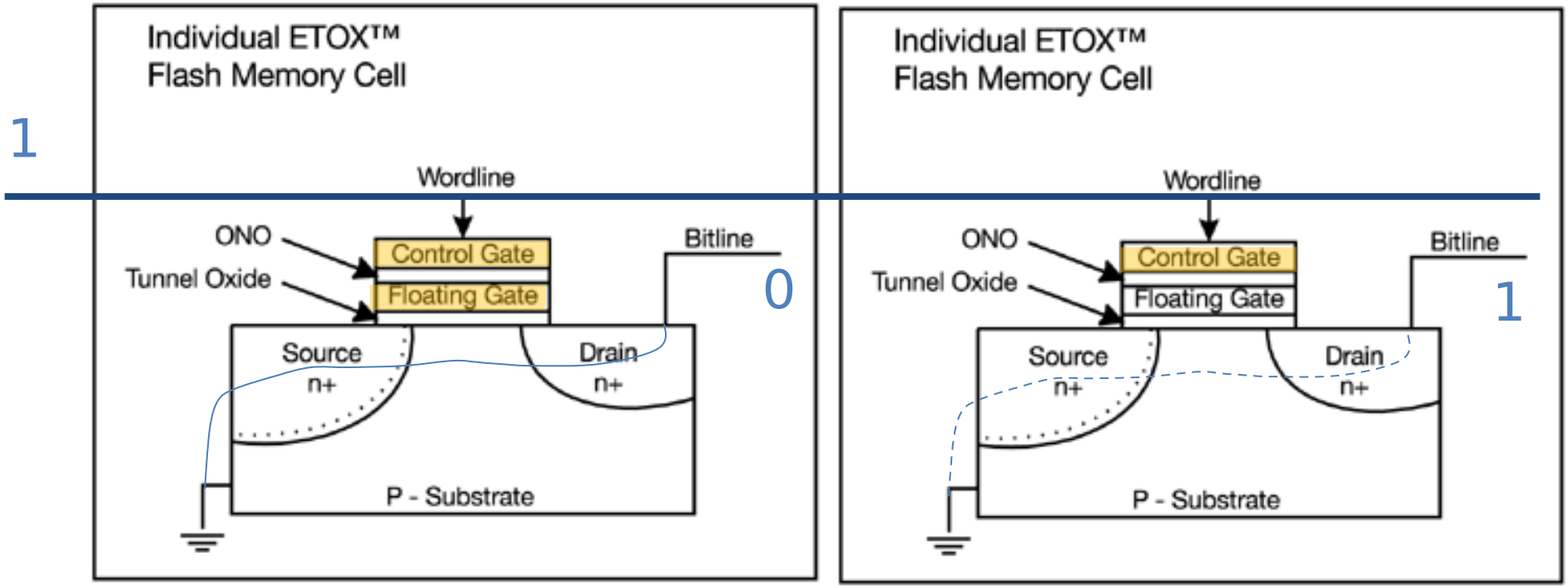
FLASH - Interface

- Looks like SRAM
  - Read
  - Write commands
- Erase is done in blocks
- Contains uCLinux kernel +file system

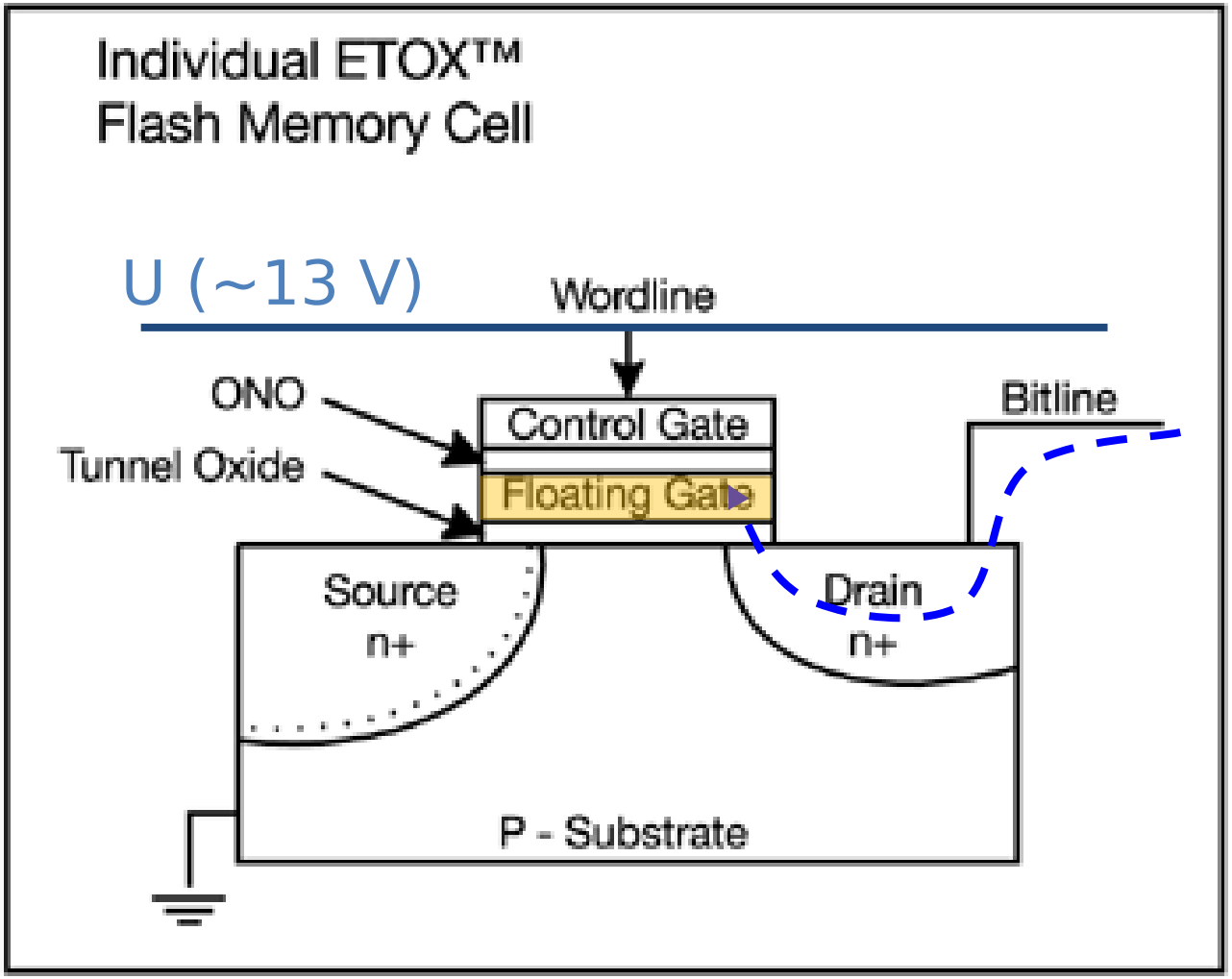
FLASH - Cell


Individual ETOX™
Flash Memory Cell
Wordline
ONO
Tunnel Oxide
Control Gate
Floating Gate
Bitline
Source
n+
Drain
n+
P - Substrate

FLASH – Read (NOR)

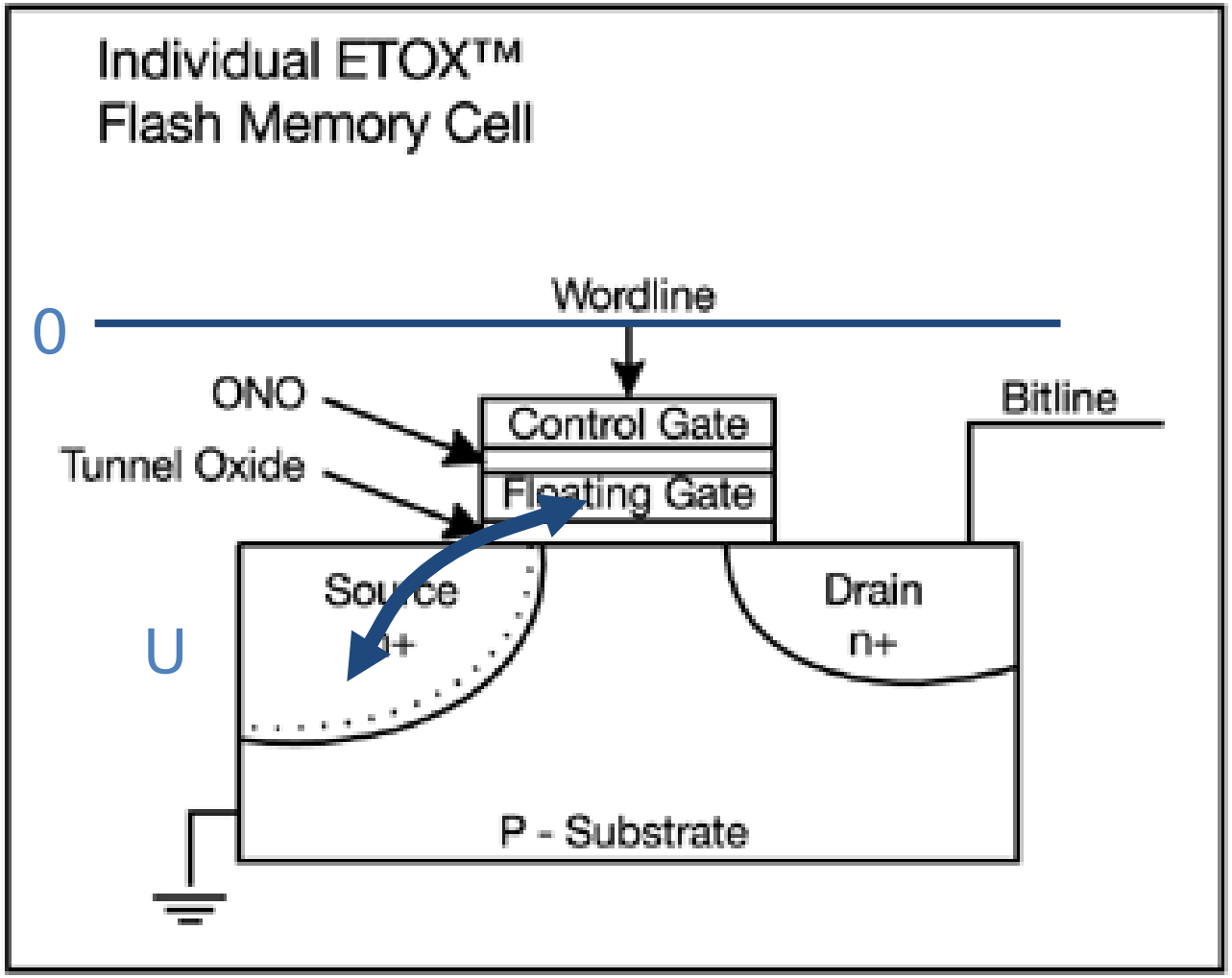


FLASH – Program (to 0)

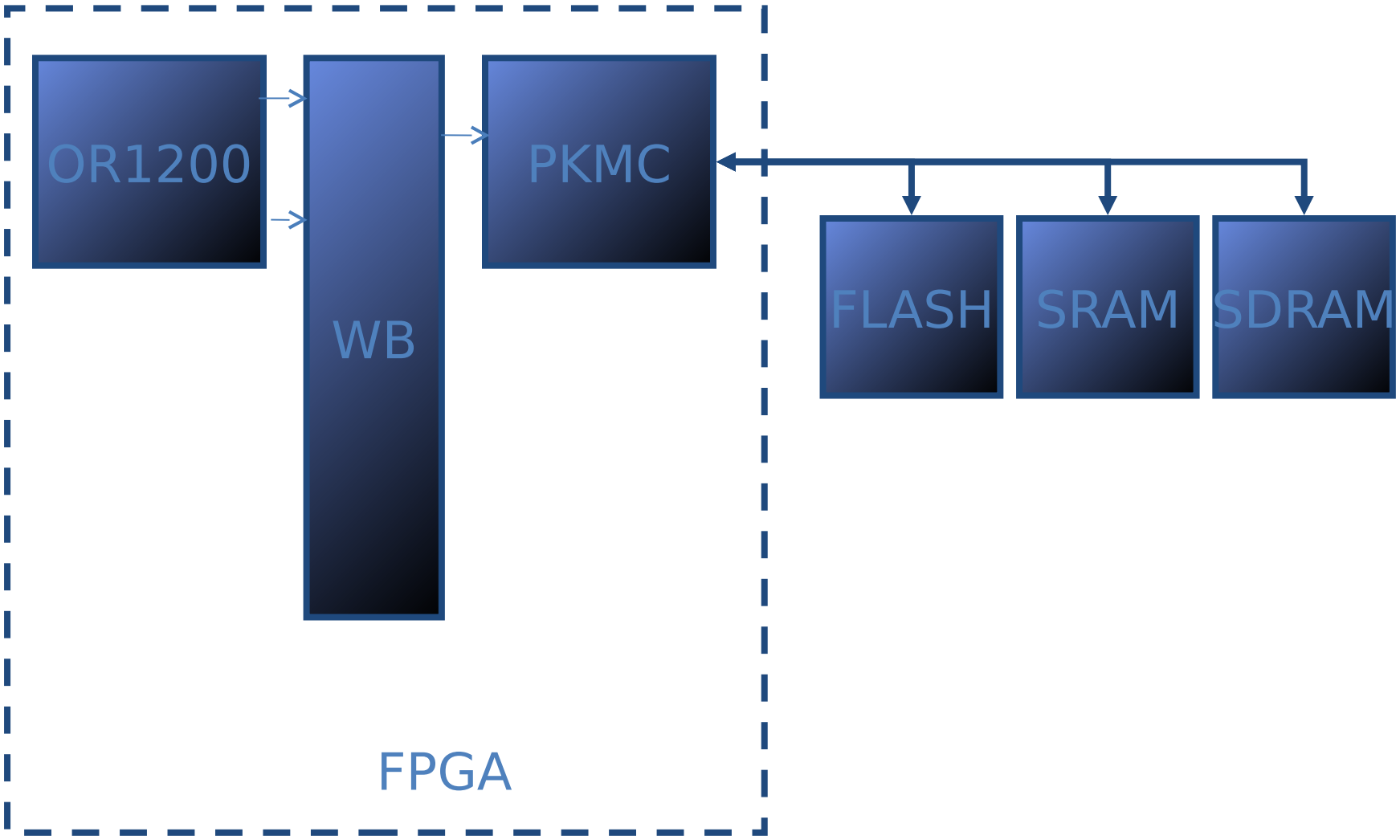


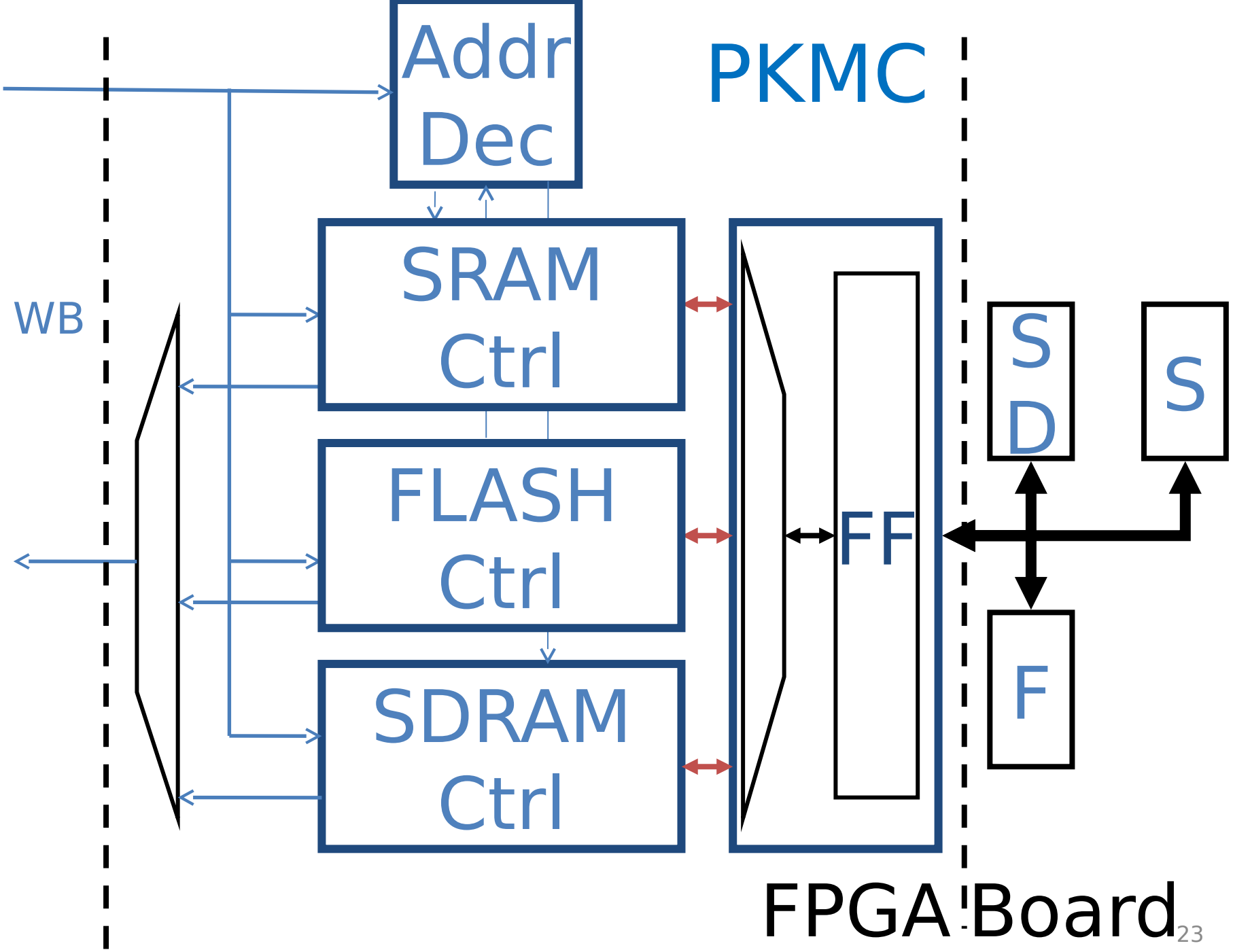
FLASH – Erase (to 1)

only blockwise

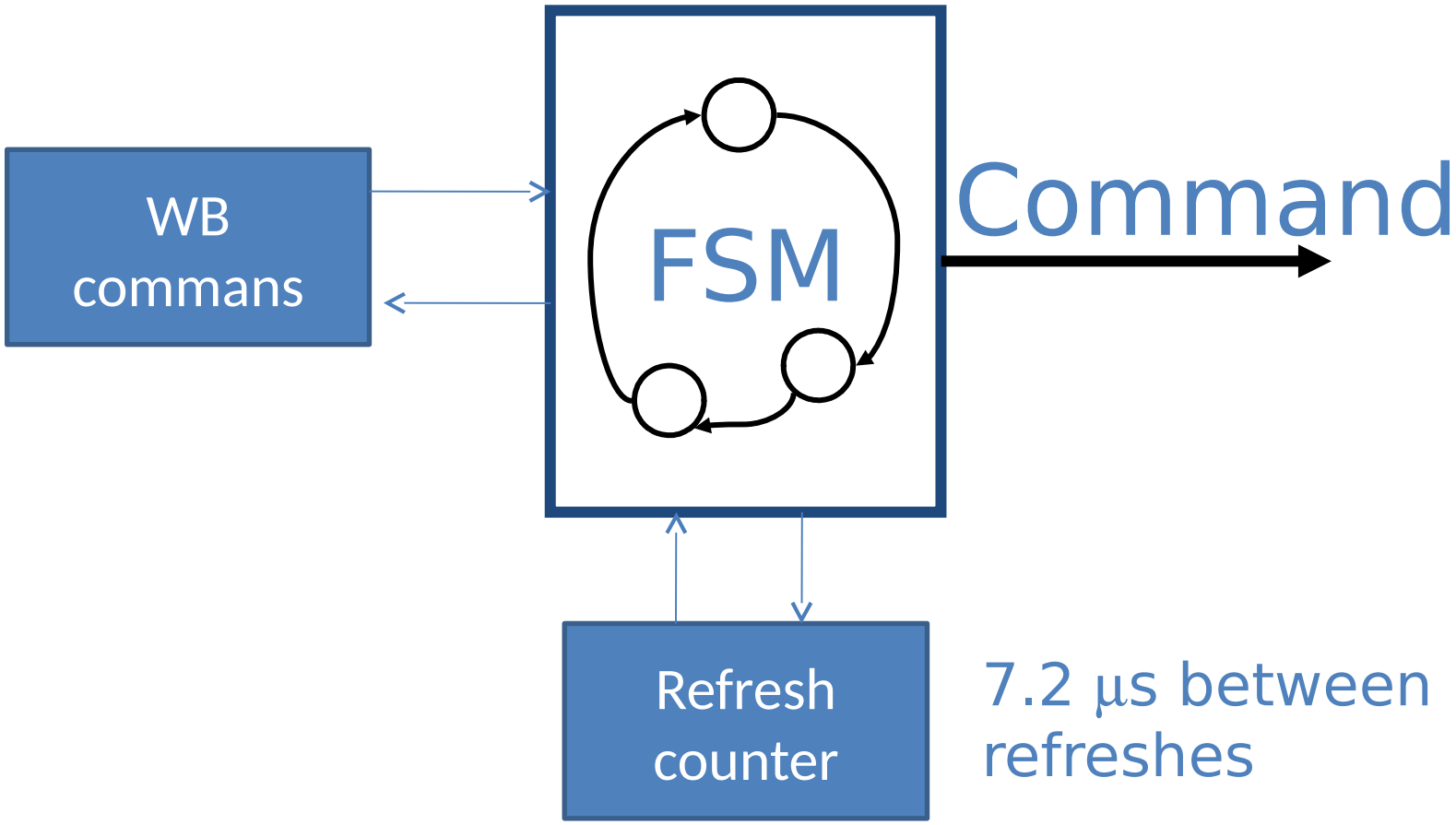


System Overwiev



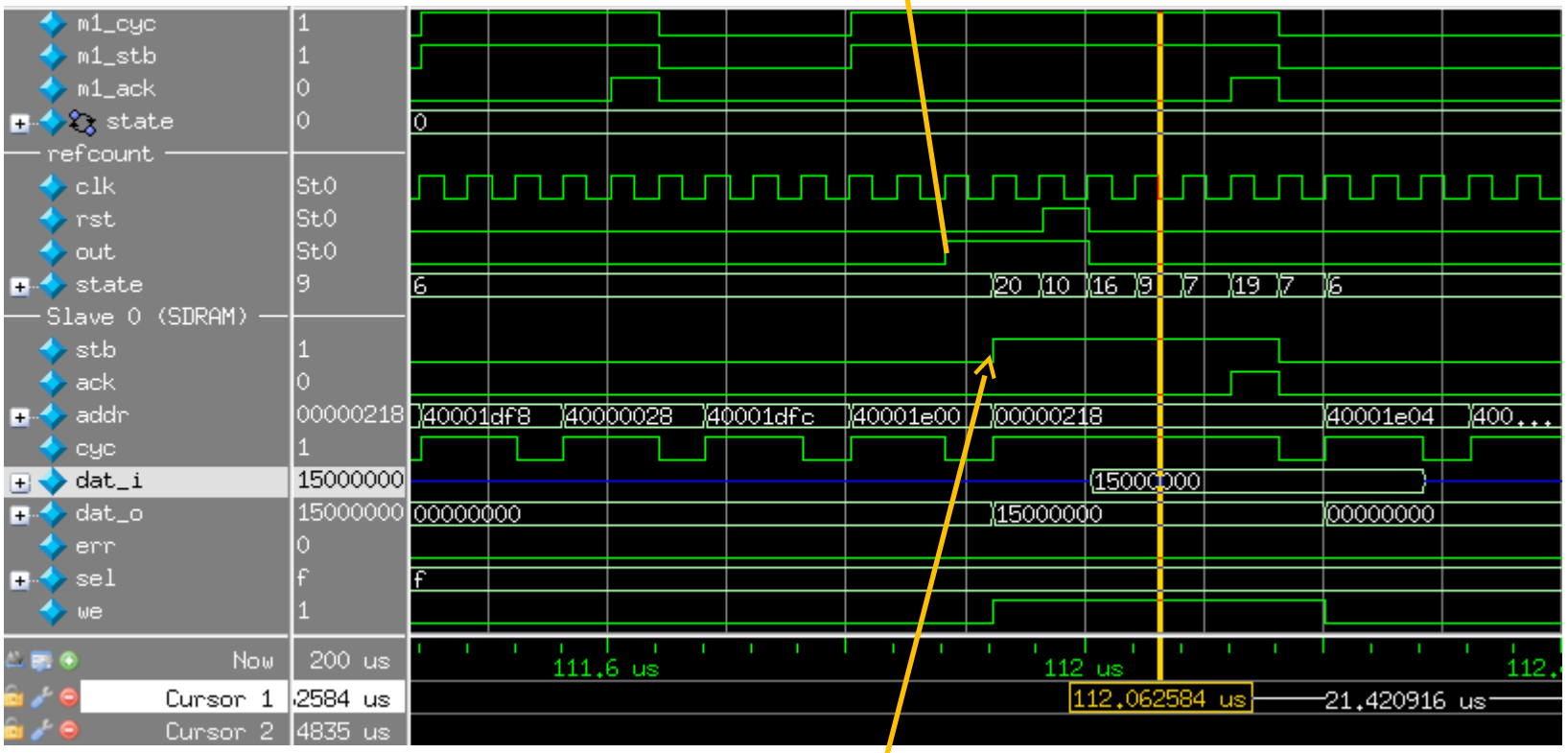

Addr
Dec
PKMC
SRAM
Ctrl
WB
FLASH
Ctrl
SDRAM
Ctrl
FF
S
D
S
F
FPGA Board

SDRAM Controller Internals



Refresh cycle

Start of refresh cycle



Start of WB cycle

Lab 4

Custom Instruction

Huffman Encoding/Decoding

1) After Q

$$\begin{bmatrix} 22 & 12 & 0 & -12 & 0 & 0 & 0 & 0 \\ 0 & 0 & -8 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

2) After zig-zag

22
12
0 4
0 0 -12
-8
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

3) After RLE

value	raw bits
05	10110
04	1100
13	100
24	0011
04	0111
F0	
F0	
D1	1
00	

Run of 0:s

magnitude

4) After Huffman

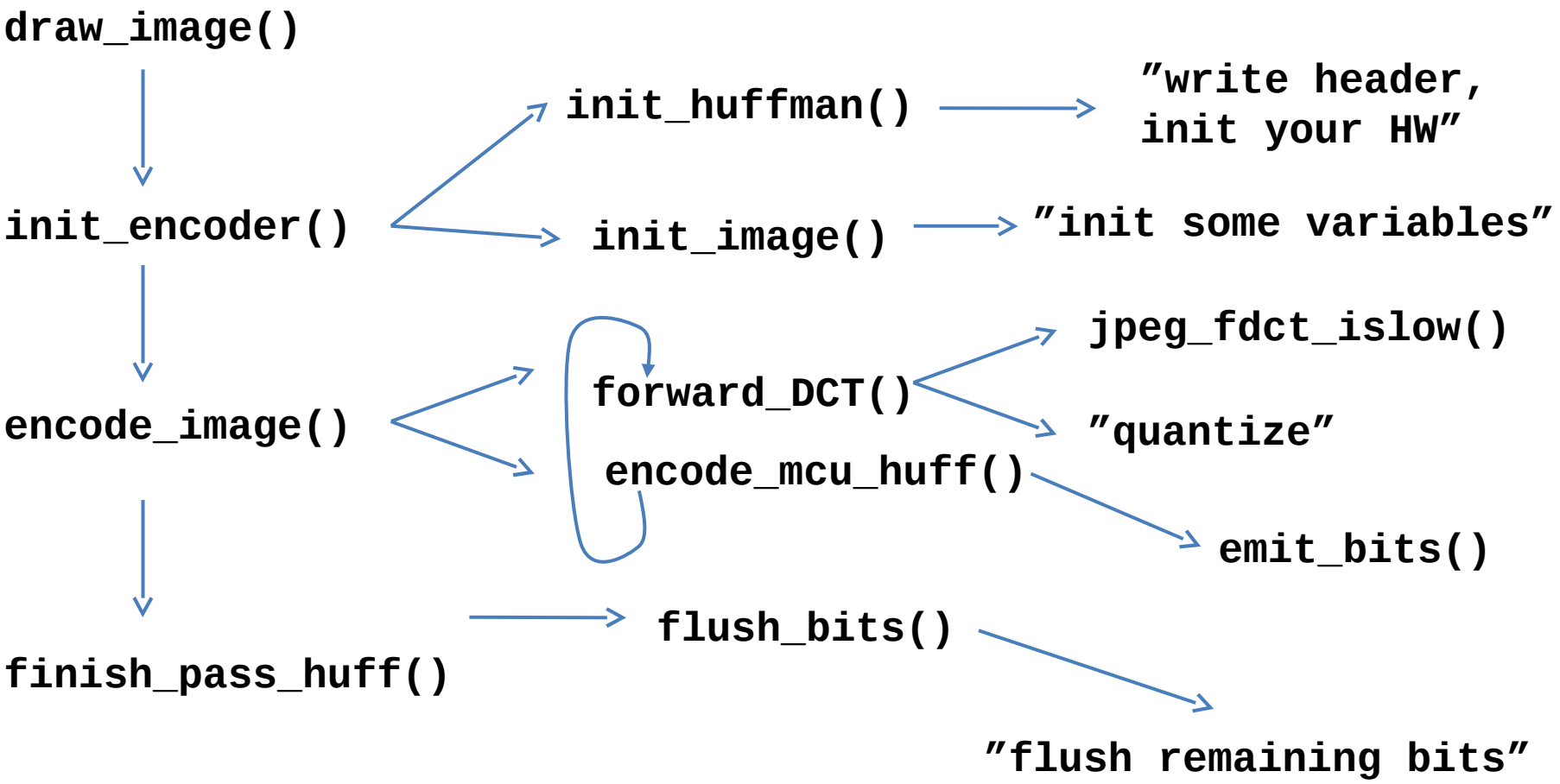
The values (bytes) are encoded by table lookup

Huffman in JFIF

- Output: 1 – 16 bits
- Encodes bytes
- 2 tables used
  - Y DC
  - Y AC

jpegfiles

jpegtest.c, jcdctmgr.c, jdct.c, jchuff.c



Emit_bits()

```
/* Only the right 24 bits of put_buffer are used; the valid bits are
 * left-justified in this part.  At most 16 bits can be passed to emit_bits
 * in one call, and we never retain more than 7 bits in put_buffer
 * between calls, so 24 bits are sufficient.
 */
static void emit_bits (unsigned int code, int size)
{
   unsigned int startcycle;

   new_put_buffer = (int) code;

// Add new bits to old bits. If at least 8 bits then write a char to buffer,
// save the rest until we get more bits.

   new_put_buffer &= (1<<size) - 1;               /* mask off any extra bits in code */
   current_buffer_bit += size;           /* new number of bits in buffer */
   new_put_buffer = new_put_buffer << (24 - current_buffer_bit);    /* align incoming bits */
   new_put_buffer = new_put_buffer | old_put_buffer;   /* and merge with old buffer contents */

   while (current_buffer_bit >= 8) {
     int c = ((new_put_buffer >> 16) & 0xFF); // Mask out the 8 bits we want
     buffer[next_buffer] = (char) c;
     next_buffer++;
     if (c == 0xFF) {         // 0xFF is a reserved code for tags, if we get image data
        buffer[next_buffer] = 0x00;     // with an FF value it has to be followed by 0x00.
        next_buffer++;
    }
    new_put_buffer <<= 8;
    current_buffer_bit -= 8;
  }
  old_put_buffer = new_put_buffer; /* update state variables */
}
```

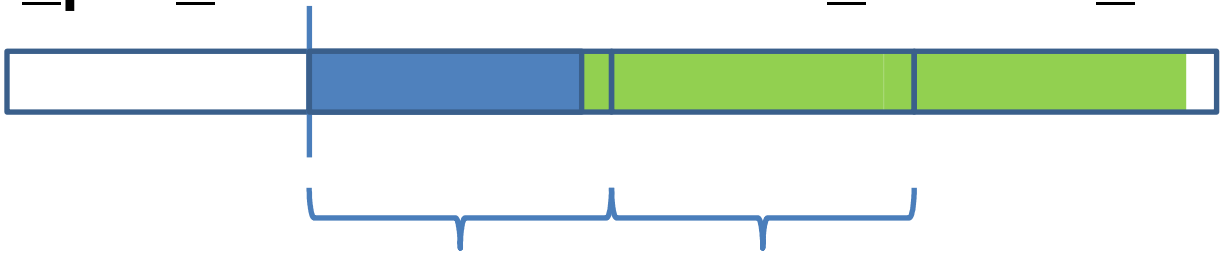
Emit_bits()

old_put_buffer **current_buffer_bit=7**

code

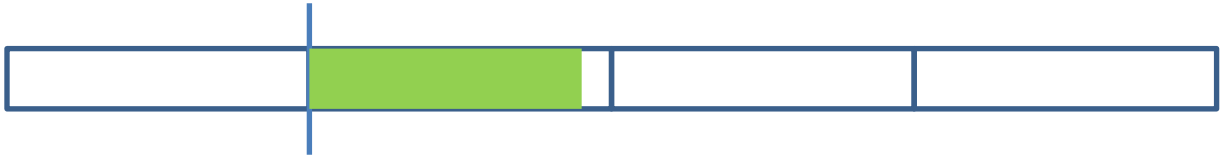
size=16

new_put_buffer **current_buffer_bit=23**



Write bytes to mem

old_put_buffer **current_buffer_bit=7**



Adding an Instruction

1. Instruction Selection
2. Hardware modification
3. Assembler modification
4. Compiler modification

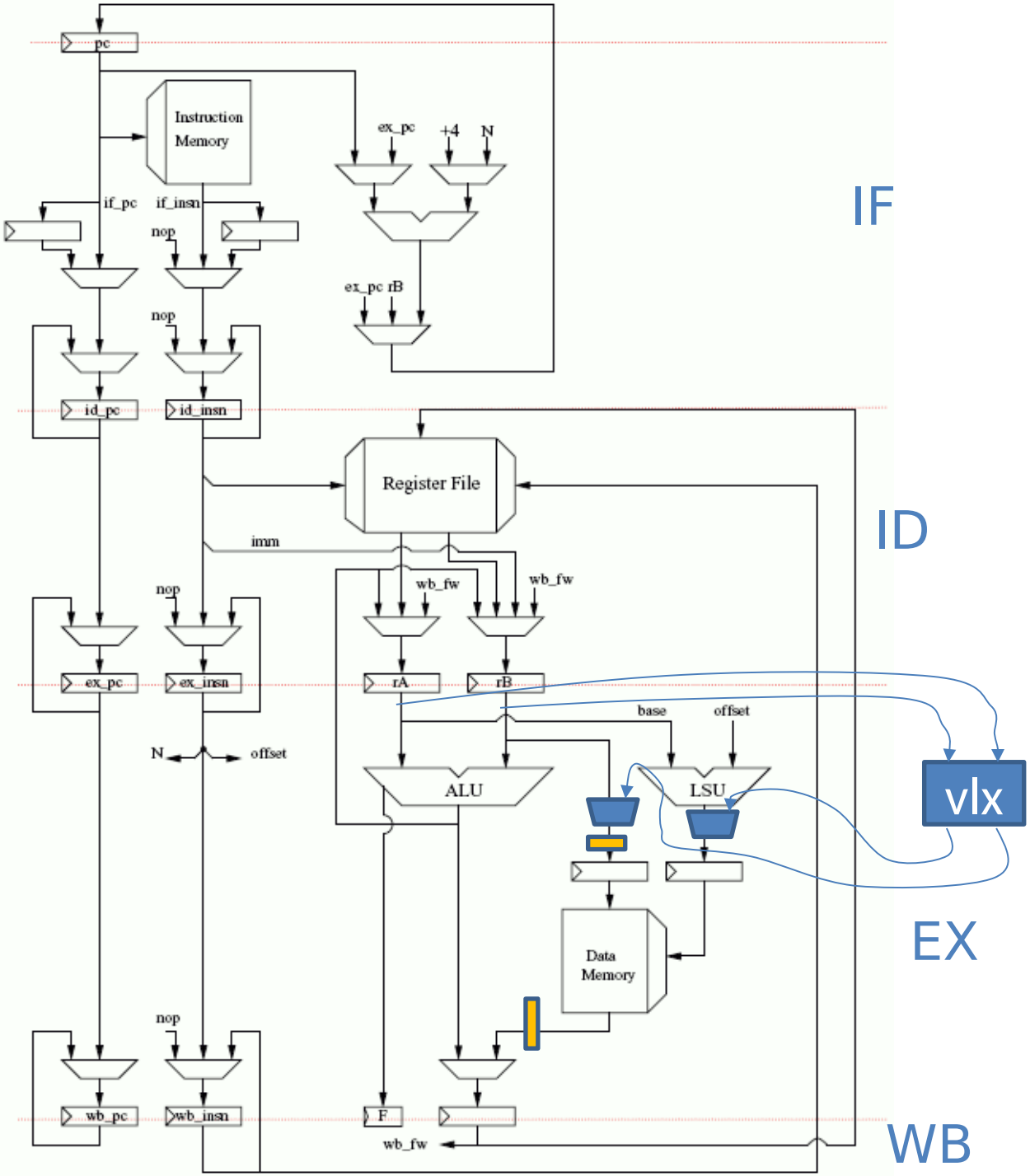
Instruction Selection

- **`l.custx`**
  - No operands
- Instructions for 64 bit
  - Not used
  - Assembler can understand
  - **`l.sd I(rA),rB`**

Hardware Modifications

- Instruction decoder modifications
  - Legal instruction
  - or1200_ctrl.v
- Special purpose register
  - New group
  - or1200_sprs.v
- Data path
  - New hardware
  - or1200_lsu.v
  - or1200_vlx_top.v

or1200 Pipeline



code size

```
l.sd (rA),rB
```

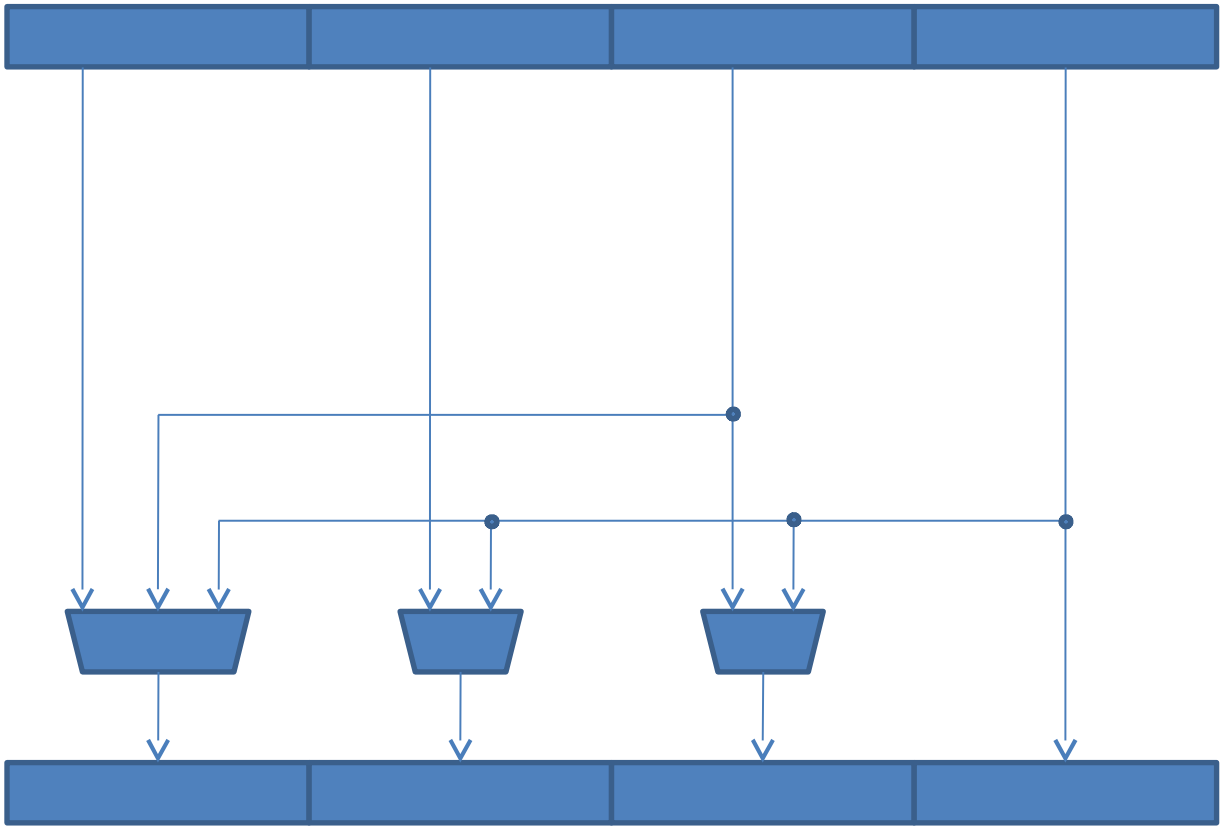
= align

or1200 Pipeline

- Remember stall

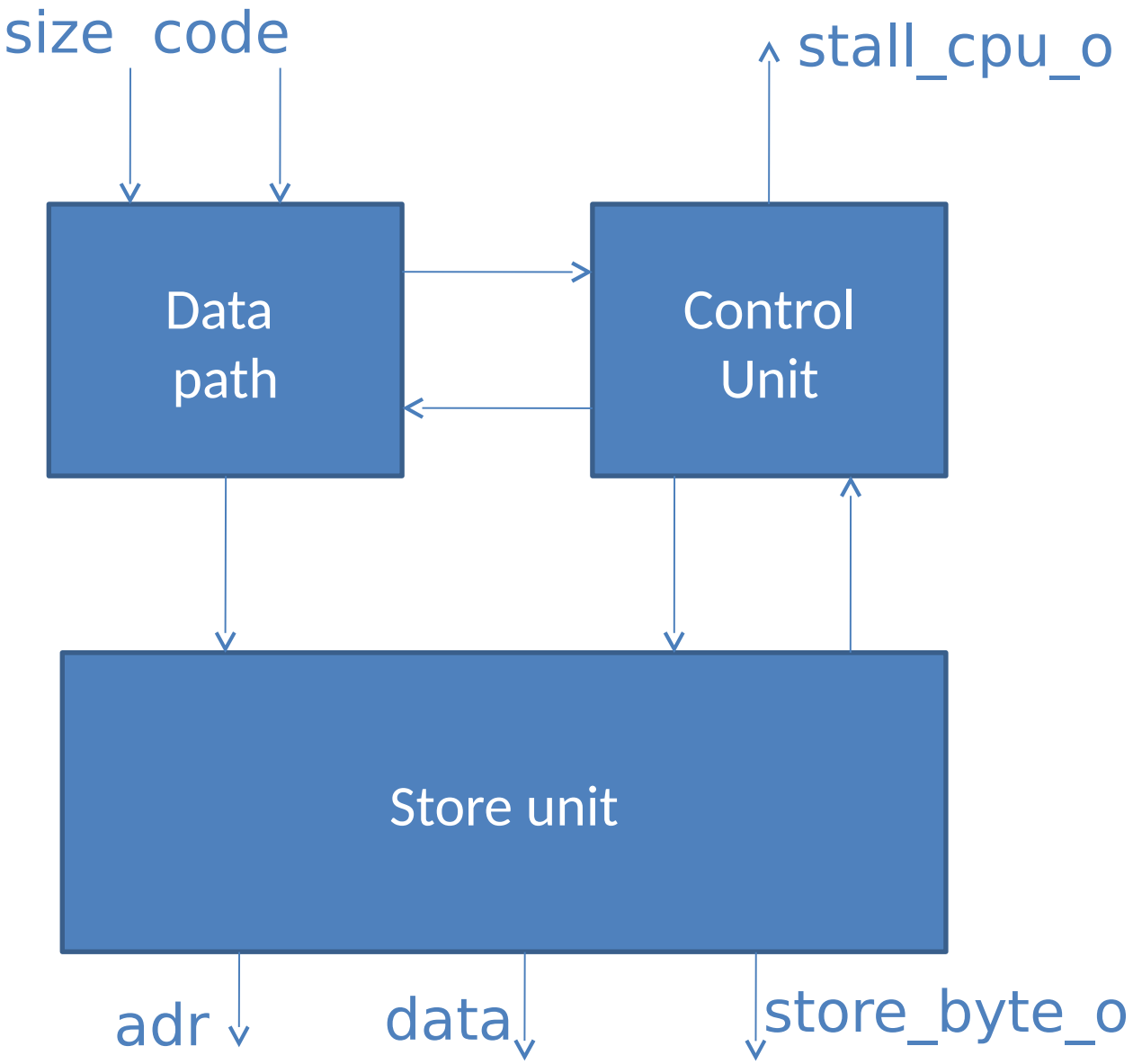
	1	2	3	4	5	6	7
IF	ld	add	sub	-			
ID/RR		ld	add	-	sub		
EX/M			ld	ld	add	sub	
W				-	ld	add	sub

Align reg2mem



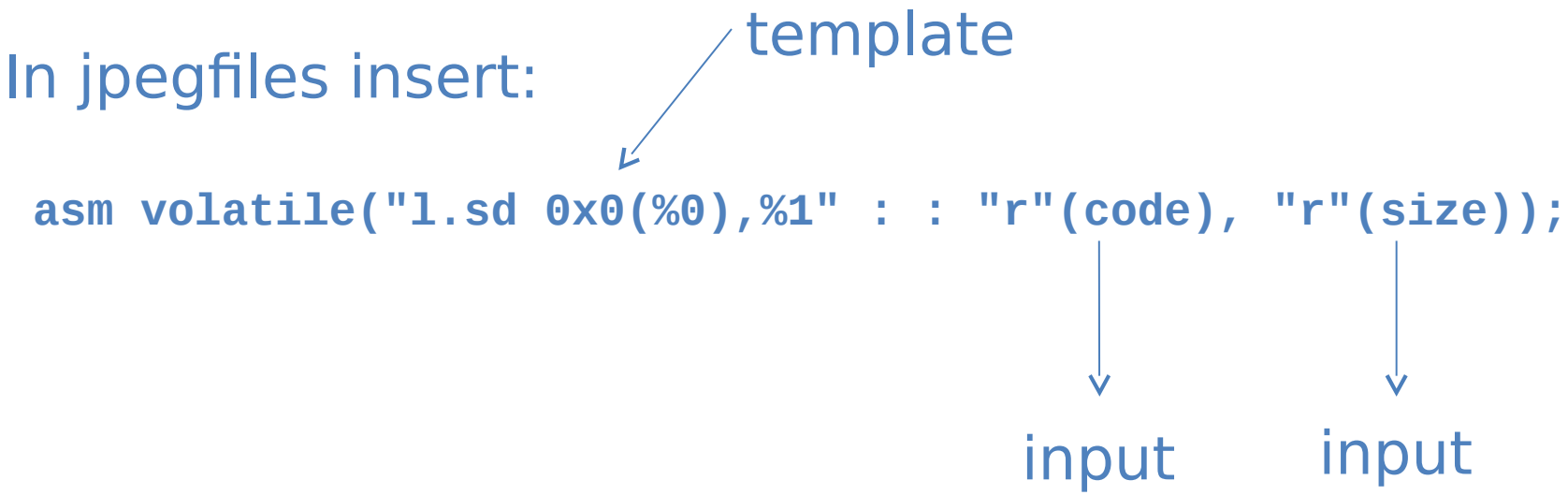
Proposed Architecture

nternal regs
(mapped as SPR)
bit_reg
bit_reg_wr_pos
vlx_addr_o



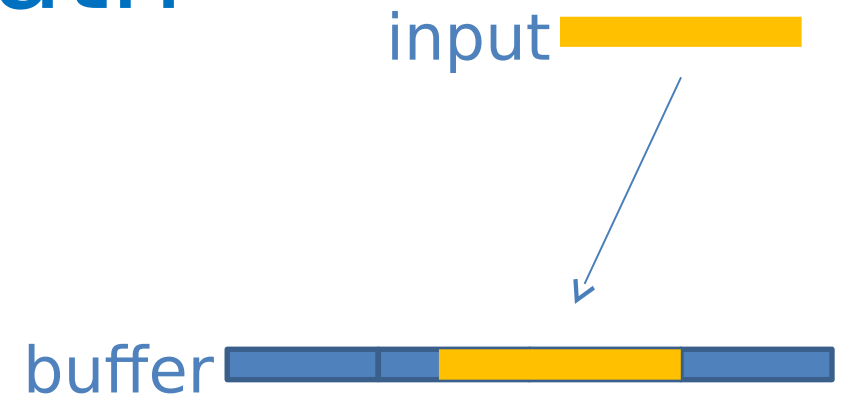
Inline asm

In jpegfiles insert:

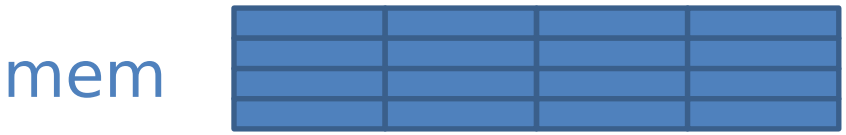


=> code and size will show up at your vlx

Data Path



- Fill buffer
  - One bit / clock cycle
  - All bits at once
- Write to mem
  - One byte
  - One 32 bit word, must be on word boundaries



Control

- May not be needed
- May be an FSM

Store Unit

- Stores the data
- 0xFF stored as 0xFF00
  - Jpeg markers
- Only byte alignment!
  - Parallel stores faster

Software

- New Assembler
  - Easy
- New Compiler
  - Hard problem for complex instructions
  - Compiler knows functions
- C
  - Inline Assembler

Instruction Usage

```
unsigned char* sb_get_buff_pos(void)
{
   unsigned char* pos;
   asm volatile("l.mfspr %0,%1,0x2":"=r"(pos):"r"(0xc000));
   return pos;
}
```

output

```
00000250 <_sb_get_buff_pos>:
 250:   9c 21 ff fc  l.addi r1,r1,0xfffffffc
 254:   d4 01 10 00  l.sw 0x0(r1),r2
 258:   9c 41 00 04  l.addi r2,r1,0x4
 25c:   a9 60 c0 00  l.ori r11,r0,0xc000
 260:   b5 6b 00 02  l.mfspr r11,r11,0x2
 264:   84 41 00 00  l.lwz r2,0x0(r1)
 268:   44 00 48 00  l.jr r9
 26c:   9c 21 00 04  l.addi r1,r1,0x4
```