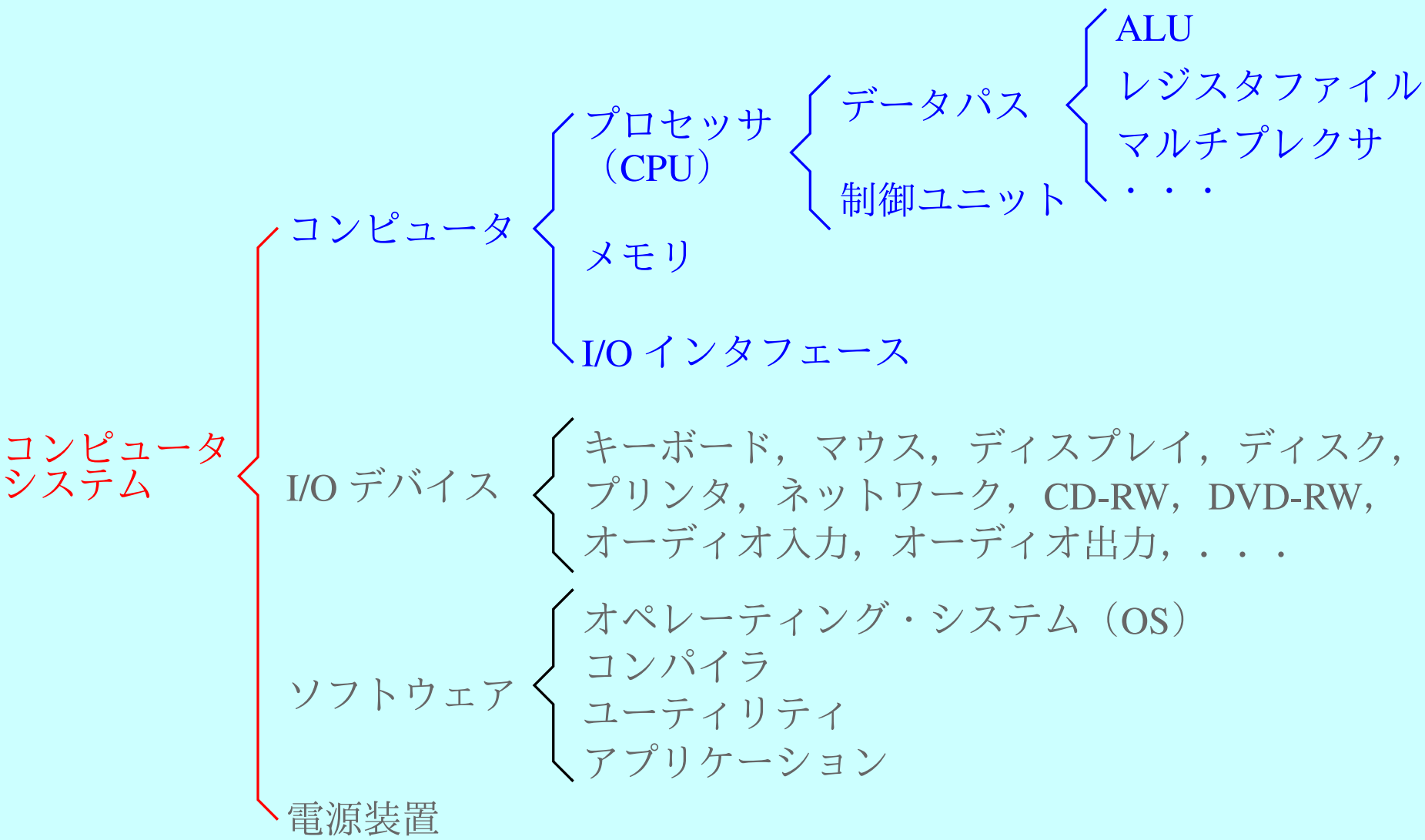
コンピュータ構成と設計（12）

入出力システム

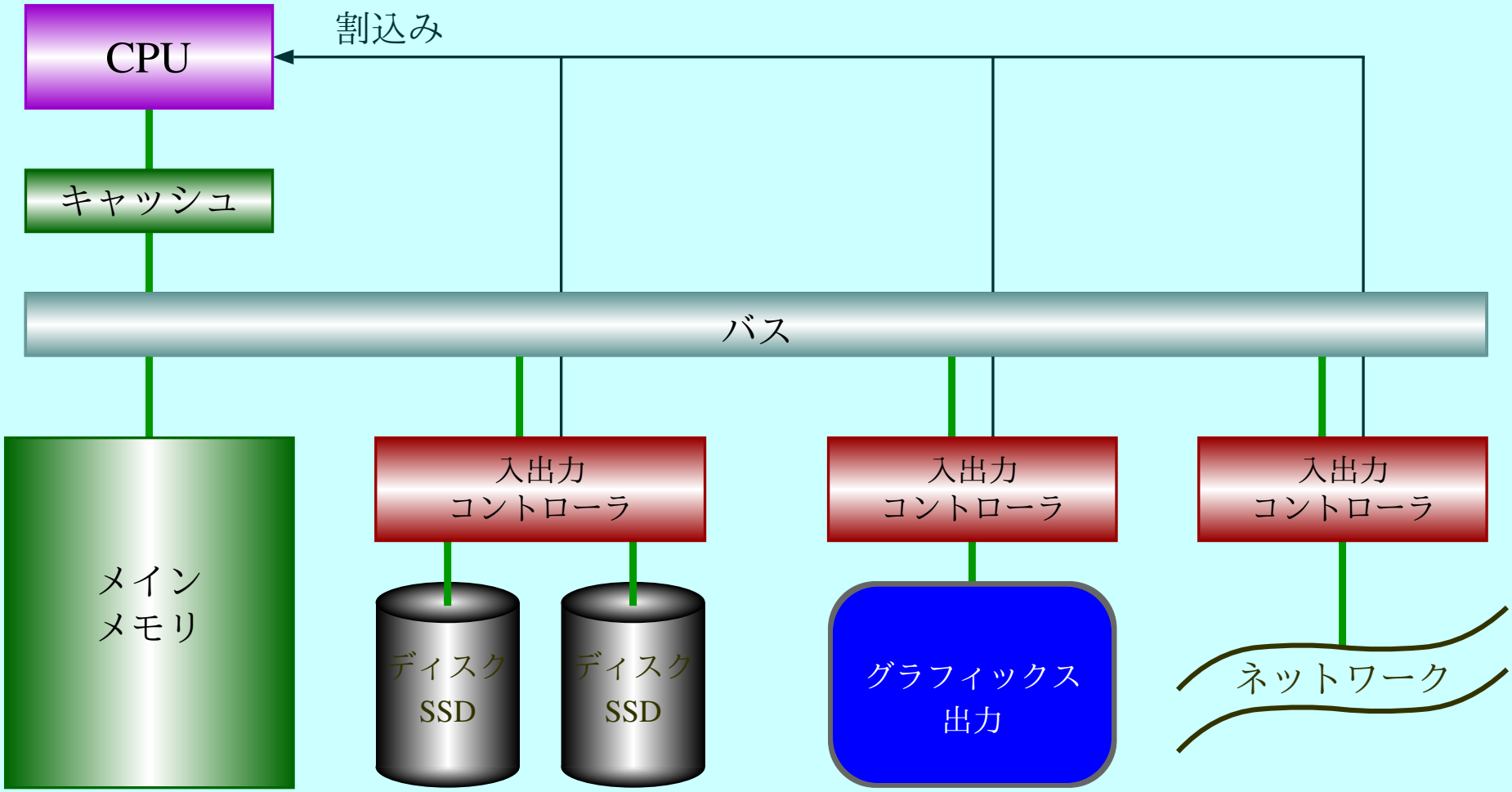
李　亜民

2024 年 12 月 19 日 (木)

コンピュータとコンピュータシステム



代表的な入出力装置


CPU
割込み
キャッシュ
バス
メイン
メモリ
入出力
コントローラ
入出力
コントローラ
入出力
コントローラ
ディスク
SSD
ディスク
SSD
グラフィックス
出力
ネットワーク

システム性能における入出力の影響

- 100 秒間実行されるベンチマークがある。そのうち 90 秒間を CPU が使い，残りの 10 秒間を入出力で使ったとする (つまり，経過時間＝ 100 秒，CPU 時間＝ 90 秒，入出力時間＝ 10 秒)
- 5 年間で，CPU は毎年 50%性能が向上するが，入出力の性能には変化がない場合，5 年後に得られるシステム全体の性能向上はどれくらいか?
- 次のことを頭に入れると，

$$
\begin{aligned}
\text{経過時間} &= \text{CPU 時間} + \text{入出力時間} \\
100 &= 90 + \text{入出力時間} \\
\text{入出力時間} &= 10 \text{ 秒}
\end{aligned}
$$

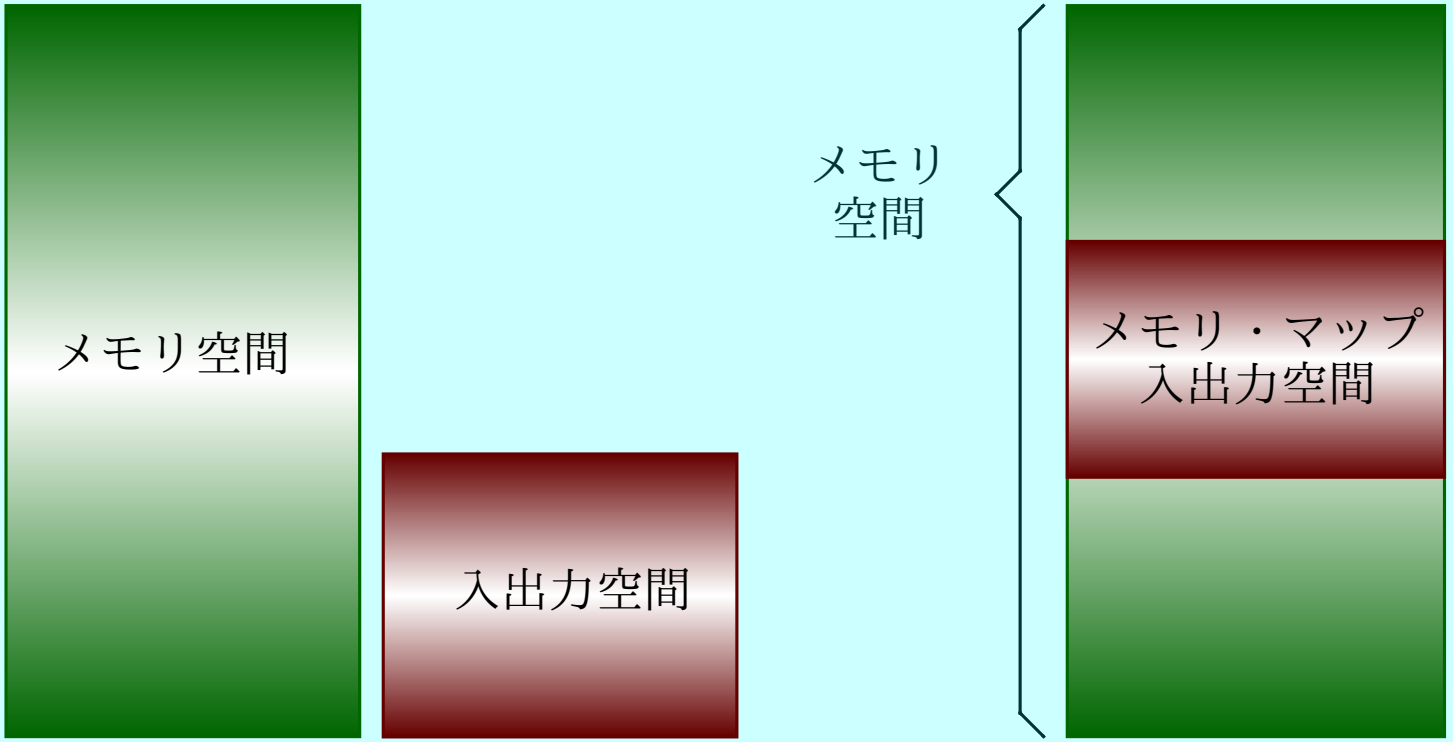
システム性能における入出力の影響

n 年後	経過時間			
	CPU 時間	入出力時間	時間	% 入出力時間
0	90 秒	10 秒	100 秒	10%
1	90/1.5 = 60 秒	10 秒	70 秒	14%
2	60/1.5 = 40 秒	10 秒	50 秒	20%
3	40/1.5 = 27 秒	10 秒	37 秒	27%
4	27/1.5 = 18 秒	10 秒	28 秒	36%
5	18/1.5 = 12 秒	10 秒	22 秒	45%

5 年間での CPU 時間の改善は，$90/12 = 7.5$

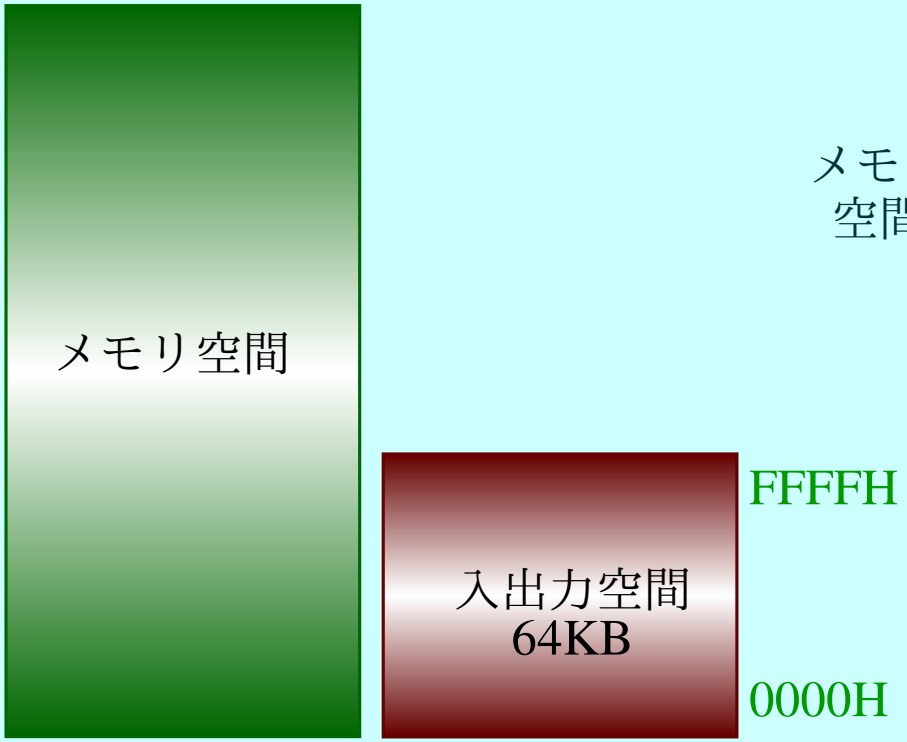
しかし，経過時間の改善は，$100/22 = 4.5$

入出力空間のタイプ

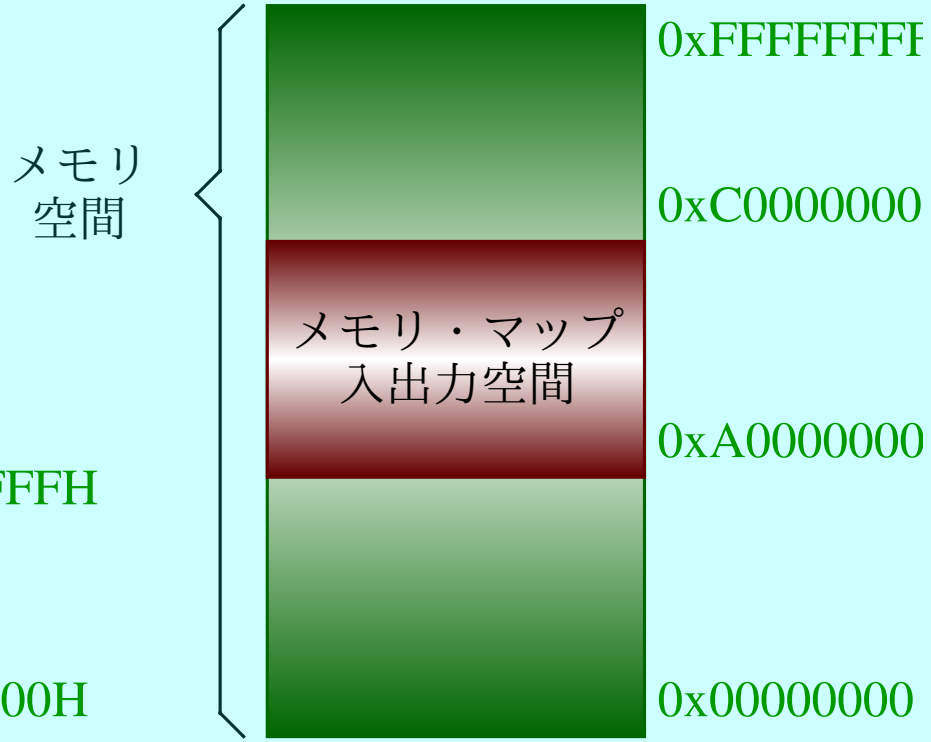


（a）別々に存在するメモリ空間と入出力空間　　（b）メモリ・マップ入出力空間

入出力空間の例



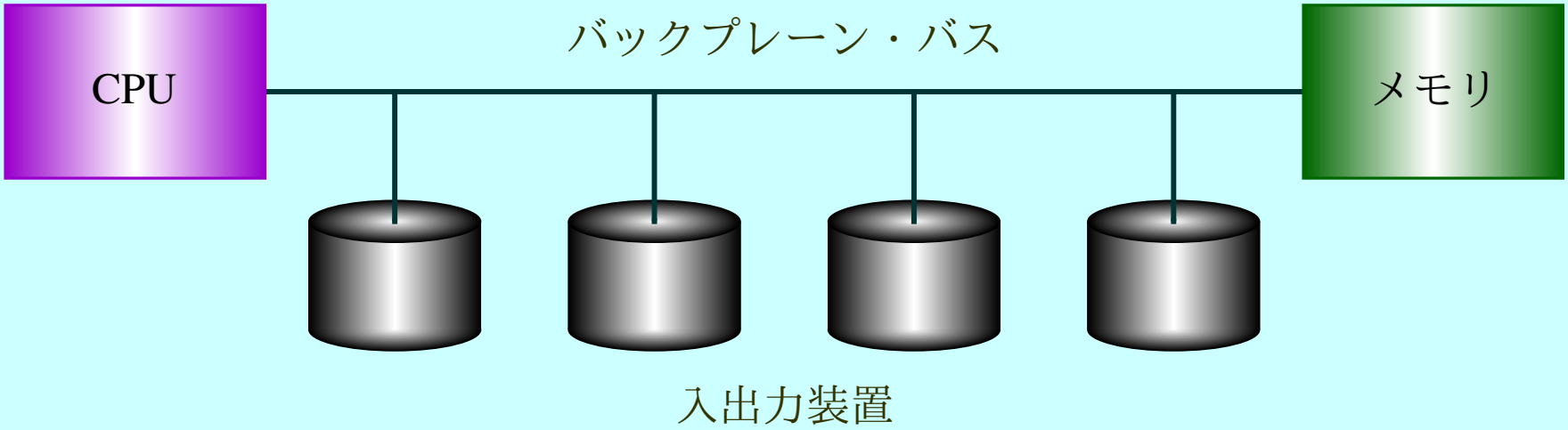
（a）Intel 8086: IN, OUT命令を使う



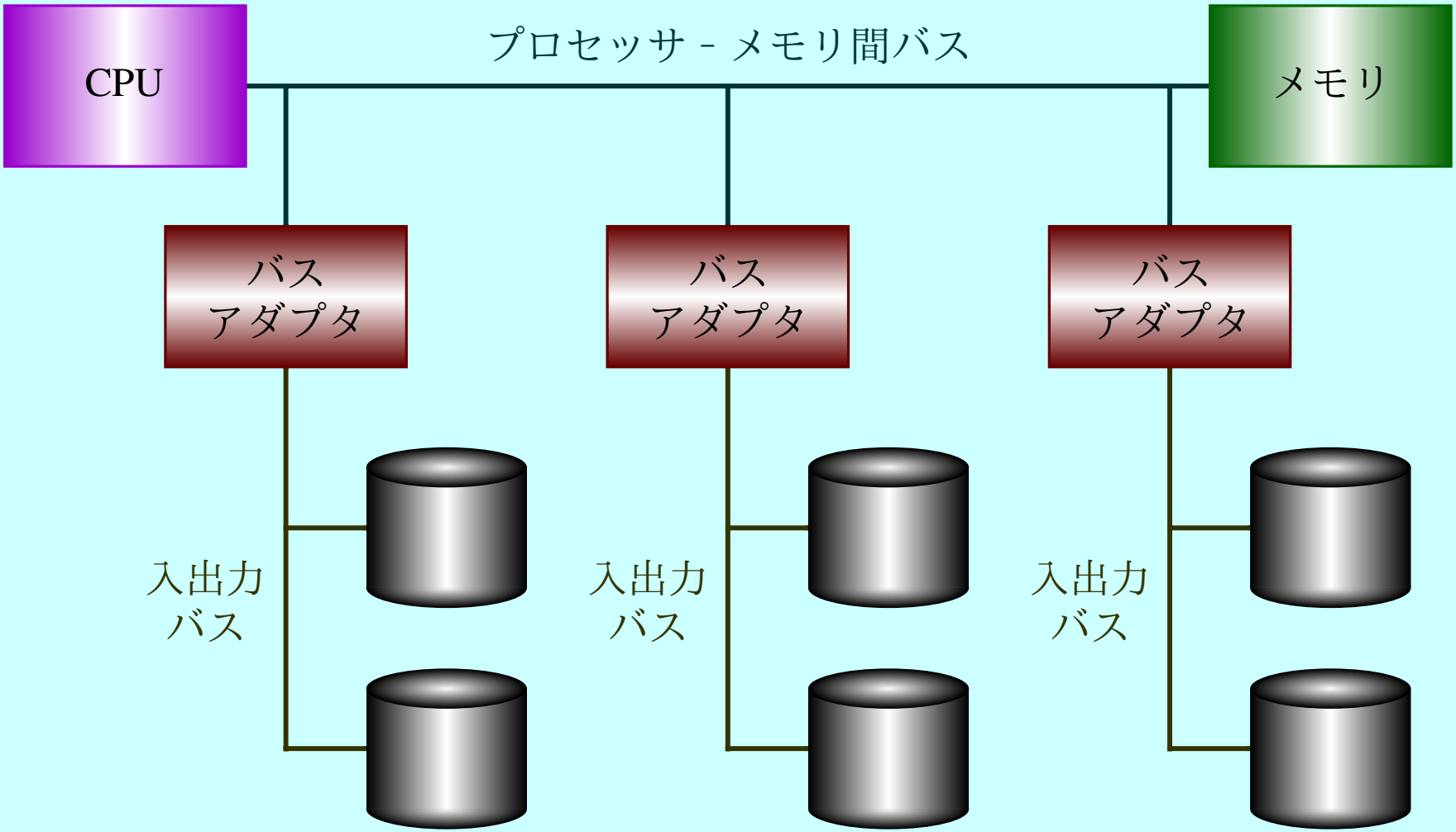
（b）RISC Load / Store命令を使う

バスのタイプ

1. プロセッサ - 主記憶間バス (Processor-Memory Bus)
2. 入出力バス (I/O Bus)
3. バックプレーン・バス (Backplane Bus)



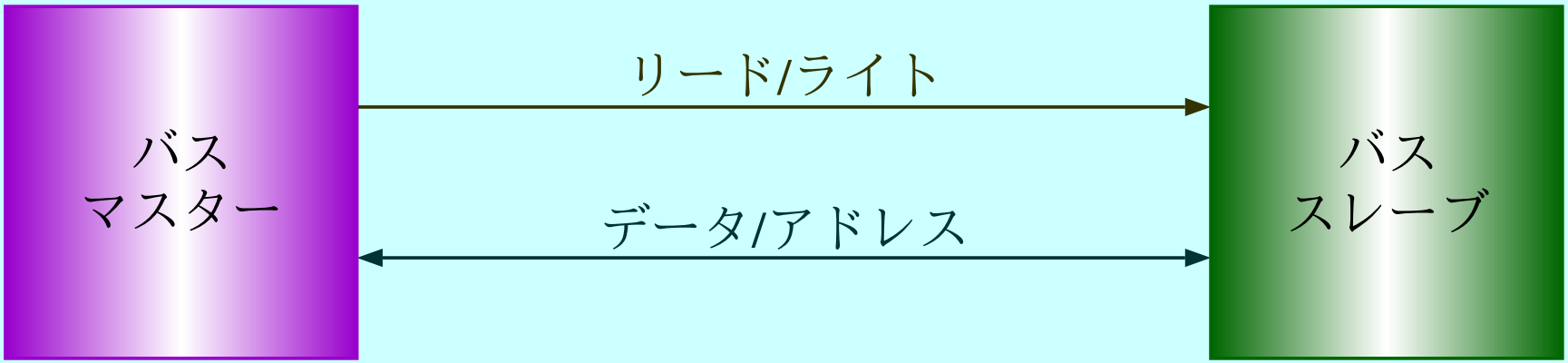
CPU - 主記憶間バスと入出力バス



バスを用いてCPUと周辺装置を接続

- バスの利点:
  - 新しい装置を加えるのが容易
  - 低コスト
- バスの欠点:
  - 通信ボトルネックが発生する
  - 最大バス速度が限られている
- バス機構
  - 制御線:
    - リード/ライト，要求，応答
  - データ線:
    - データ，アドレス

バス・マスターとスレーブ



バス・マスター:

1. アドレスとリード/ライトを送信
2. ライトならばデータを送信

バス・スレーブ:

1. リードならばデータを送信
2. ライトならばデータを受信

同期式バスと非同期式バス

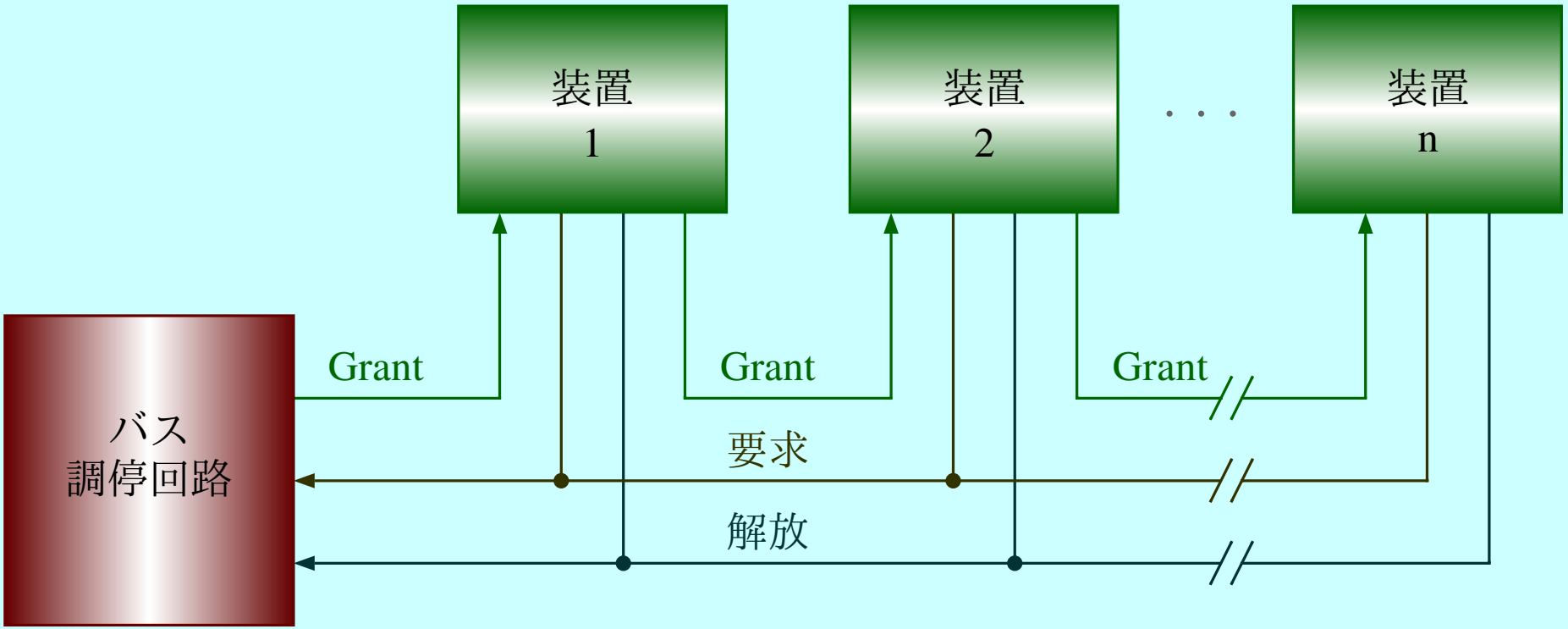
- バスの通信方式
  - 非同期式
  - 同期式
- 非同期式バス
  - クロック制御ではない
    - ★ クロック・スキューに煩わされることなくバス長を伸ばすことができる
  - ハンドシェーク型プロトコルが必要
  - 多種多様な装置に対応できる

同期式バス

- 制御線の中にクロック用の線が組み込まれている
- クロックを基準とした固定的な通信プロトコル
- 利点
  - 論理が小規模
  - 非常に高速
- 欠点
  - 同じバスに接続される装置は同じクロック周波数で動作しなければならない
  - クロック・スキューの問題が発生するため，バス長を長くできない

バス調停機構 — デイジーチェーン方式

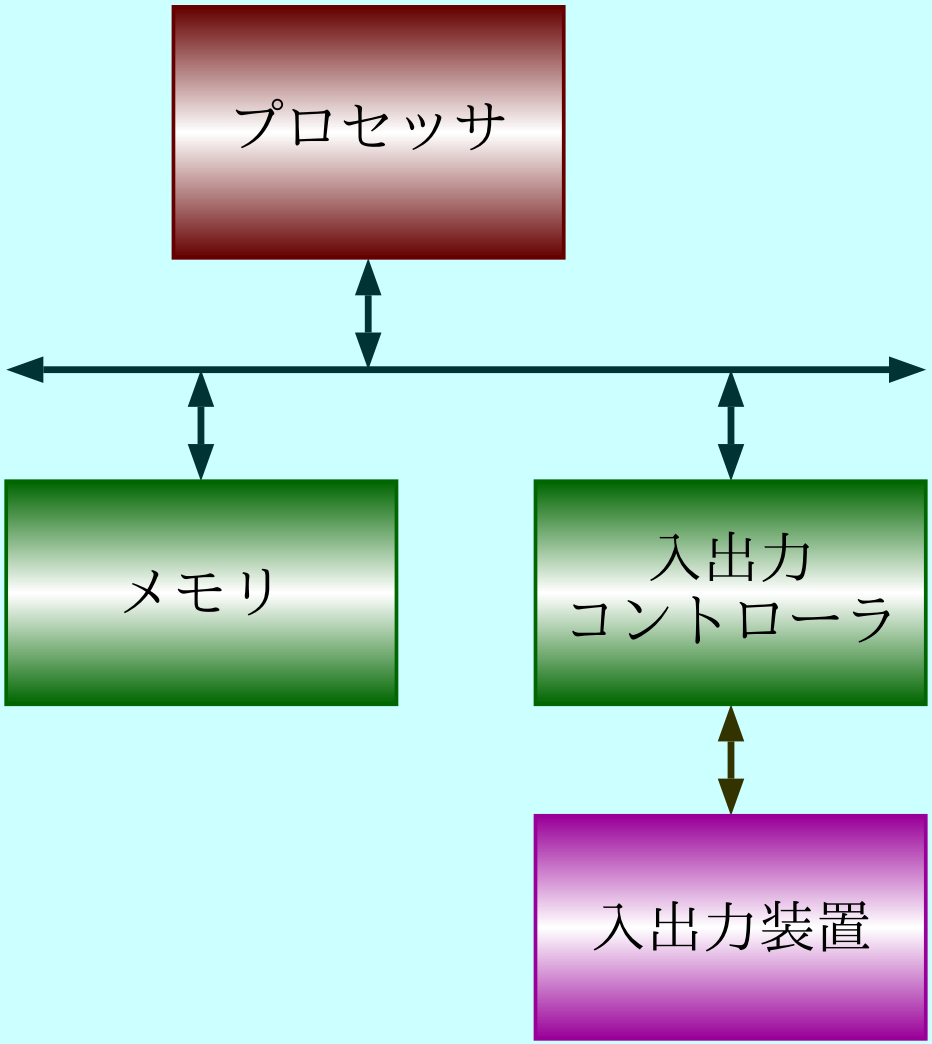
デイジーチェーン: Daisy Chain

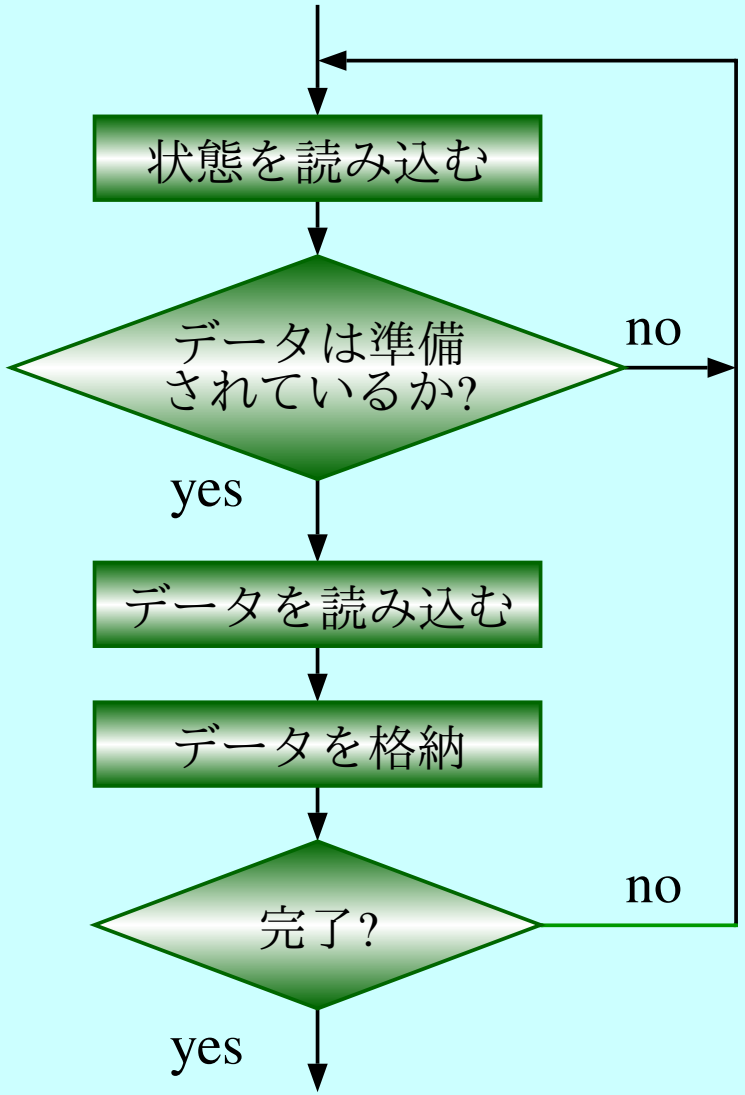


ポーリングと割込み

- 入出力装置とプロセッサの通信
  - ポーリング (Polling)
    - 入出力装置は状態レジスタに情報を置く
    - プロセッサは定期的にその状態レジスタをチェックする
  - 割込み (Interrupt)
    - 入出力装置はプロセッサの対応を必要とするやいなや，プロセッサが現在実行している処理に割込んで，要求を通知する

入出力ポーリング (Polling)

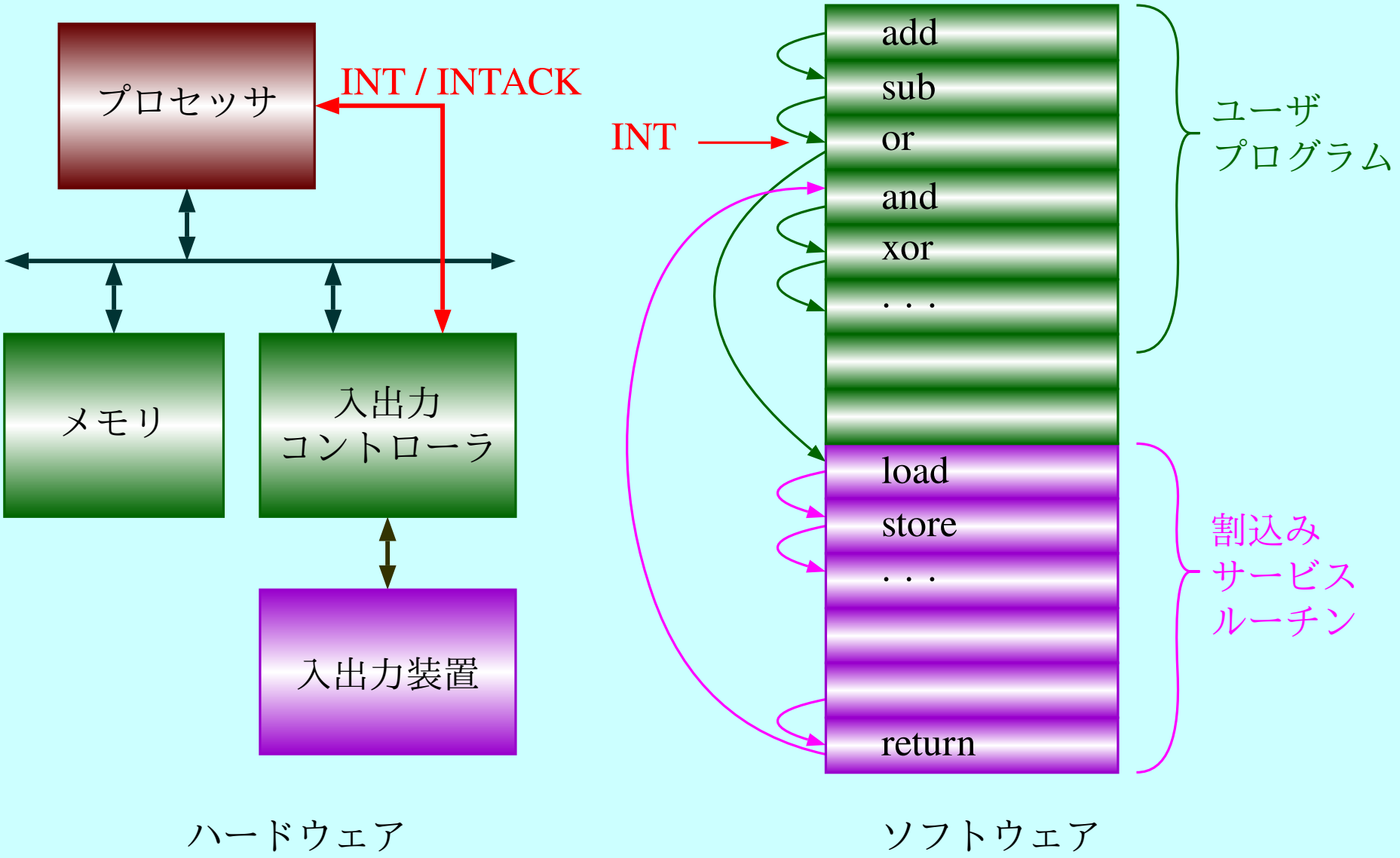




入出力ポーリング (Polling)

- ポーリングのステップ
  - 入出力装置は状態レジスタに情報を置く
  - プロセッサは定期的にその状態レジスタをチェックする
- ポーリングの利点
  - 単純: プロセッサは完全に制御下にありすべての処理を行う
- ポーリングの欠点
  - オーバーヘッドが大きく，多くの CPU 時間が無駄に使われてしまう

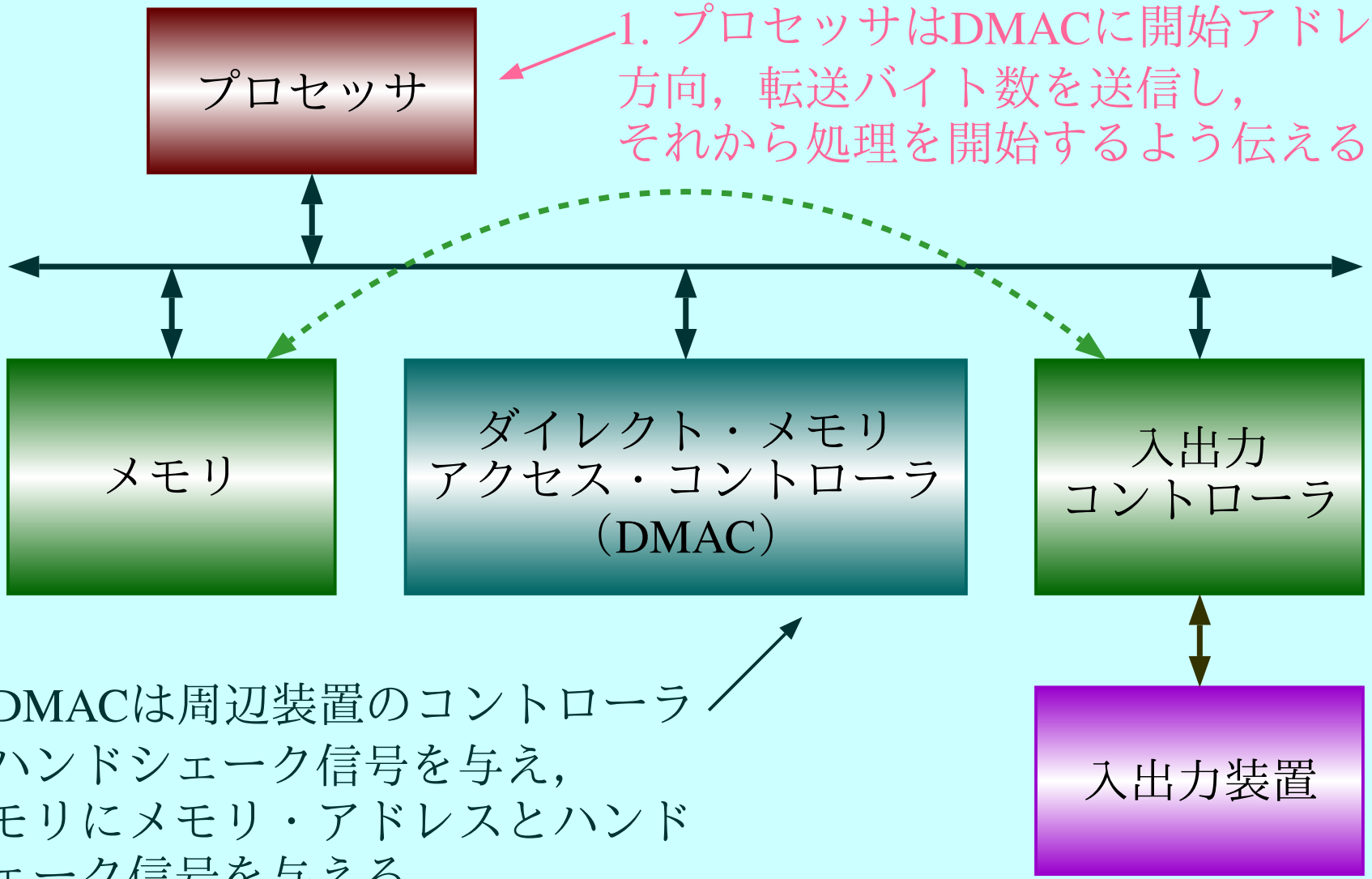
入出力割込み (Interrupt)



入出力割込み (Interrupt)

- 割込み
  - 入出力装置はプロセッサの対応を必要とするやいなや，プロセッサが現在実行している処理に割込んで，要求を通知する
- 割込みの利点
  - ユーザ・プログラムが停止するのは実際に転送されている間のみ
- 割込みの欠点
  - 特別なハードウエアが必要

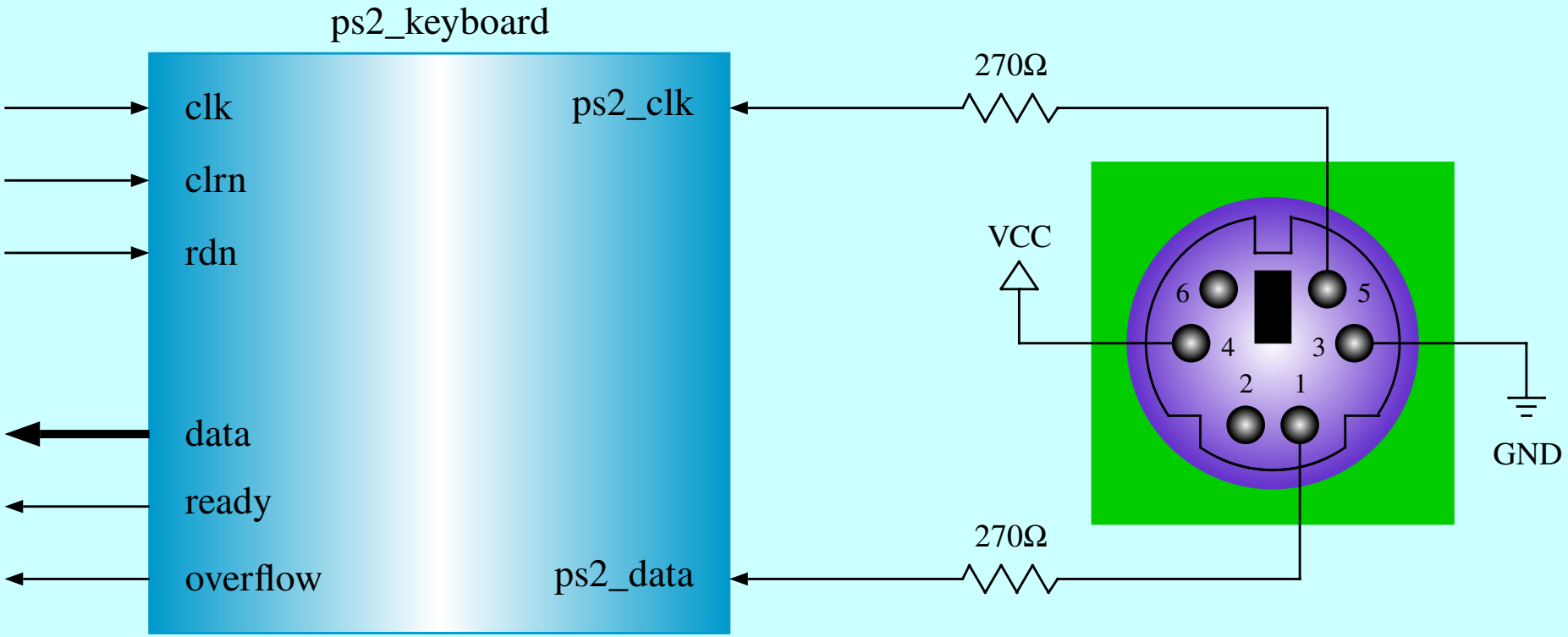
ダイレクト・メモリ・アクセス (DMA)

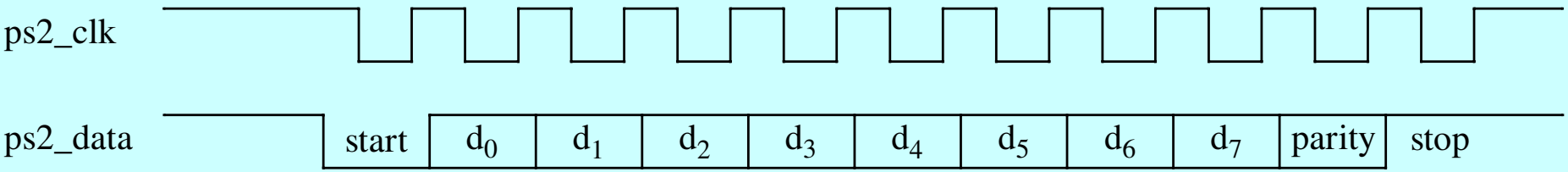


ダイレクト・メモリ・アクセス (DMA)

- DMA (Direct Memory Access) の目的
  - プロセッサを介さずに入出力装置とやり取りする
- DMAC — ダイレクト・メモリ・アクセス・コントローラ (Direct Memory Access Controller)
  - CPU の範囲外
  - バス上でマスターとして活動する
  - プロセッサを介さずにデータ・ブロックを入出力装置からメモリへ，または，メモリから入出力装置へ転送する

PS/2 Keyboard/Mouse





ps2_keyboard.v

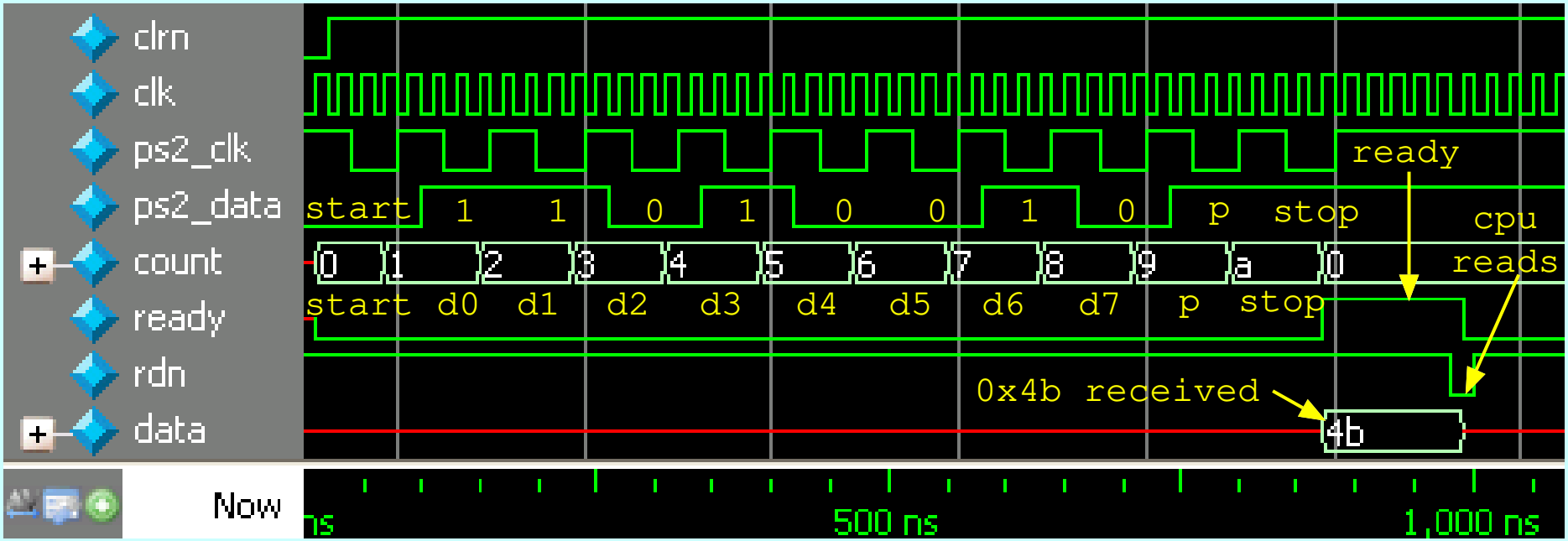
```
‘timescale 1ns/1ns
module ps2_keyboard (clk,clrn,ps2_clk,ps2_data,rdn,data,ready,overflow);
    input        clk, clrn;                                  // 50 MHz
    input        ps2_clk, ps2_data;                          // ps2 clock, ps2 data
    input        rdn;                                        // read, active low
    output [7:0] data;                                       // 8-bit code
    output       ready;                                      // code ready
    output reg   overflow;                                   // fifo overflow
    reg    [9:0] buffer;                                     // ps2_data bits
    reg    [7:0] fifo[7:0];                                  // circular fifo
    reg    [3:0] count;                                      // count ps2_data bits
    reg    [2:0] w_ptr,r_ptr;                                // fifo w/r pointers
    reg    [3:0] ps2_clk_sync;                               // for detecting falling edge
    always @ (posedge clk)
        ps2_clk_sync <= {ps2_clk_sync[2:0],ps2_clk};         // shift ps2_clk left
    wire sampling = ps2_clk_sync[3] &                        // 1
                    ps2_clk_sync[2] &                        // 1
                   ~ps2_clk_sync[1] &                        // 0
                   ~ps2_clk_sync[0];                         // 0: had a falling edge
    always @ (posedge clk) begin
        if (!clrn) begin                                     // on reset
            count    <= 0;                                   // clear count
            w_ptr    <= 0;                                   // clear w_ptr
            r_ptr    <= 0;                                   // clear r_ptr
            overflow <= 0;                                   // clear overflow
```

ps2_keyboard.v

```
        end else if (sampling) begin                    // if sampling
            if (count == 4'd10) begin                   // if got one frame
                if ((buffer[0] == 0) && (ps2_data) && (^buffer[9:1])) begin
                    if ((w_ptr + 3'b1) != r_ptr) begin
                        fifo[w_ptr] <= buffer[8:1];
                        w_ptr <= w_ptr + 3'b1;          // w_ptr++
                    end else begin
                        overflow <= 1;                  // overflow
                    end
                end
                count <= 0;                             // for next frame
            end else begin                              // else
                buffer[count] <= ps2_data;              // store ps2_data
                count <= count + 4'b1;                  // count++
            end
        end
        if (!rdn && ready) begin                        // on cpu read
            r_ptr <= r_ptr + 3'b1;                      // r_ptr++
            overflow <= 0;                              // clear overflow
        end
    end

    assign ready = (w_ptr != r_ptr);                    // fifo is not empty
    assign data  = fifo[r_ptr];                         // code byte
endmodule
```

Simulation



The keyboard sends Scan Code to the keyboard controller

Scan Code

- The Scan Code consists of Make Code and Break Code
- For example, if you pressed a key ‘L’ and release it immediately, the keyboard controller will give you a scan code of three bytes: 0x4b 0xf0 0x4b.
- If you pressed a key ‘Left Shift’, then ‘L’, and release ‘L’ and then ‘Left Shift’, the keyboard controller will give you the scan codes of six bytes: 0x12 0x4b 0xf0 0x4b 0xf0 0x12. That is,

```
0x4b 0xf0 0x4b                   : `l' (ASCII 0x6c)
0x12 0x4b 0xf0 0x4b 0xf0 0x12 : `L' (ASCII 0x4c)
```

Some Keys' Scan Codes

Key	MakeCode	BreakCode
[1 !] ...	16 ... 16	f0 16
[2 "] ...	1e ... 1e	f0 1e
[3 #] ...	26 ... 26	f0 26
[4 $] ...	25 ... 25	f0 25
[5 %] ...	2e ... 2e	f0 2e
[6 &] ...	36 ... 36	f0 36
[7 '] ...	3d ... 3d	f0 3d
[8 (] ...	3e ... 3e	f0 3e
[9)] ...	46 ... 46	f0 46
[0] ...	45 ... 45	f0 45
[A] ...	1c ... 1c	f0 1c
[B] ...	32 ... 32	f0 32
[C] ...	21 ... 21	f0 21
[D] ...	23 ... 23	f0 23
[E] ...	24 ... 24	f0 24
[F] ...	2b ... 2b	f0 2b
[G] ...	34 ... 34	f0 34

Some Keys' Scan Codes

[H]	...	33	...	33	f0	33
[I]	...	43	...	43	f0	43
[J]	...	3b	...	3b	f0	3b
[K]	...	42	...	42	f0	42
[L]	...	4b	...	4b	f0	4b
[M]	...	3a	...	3a	f0	3a
[N]	...	31	...	31	f0	31
[O]	...	44	...	44	f0	44
[P]	...	4d	...	4d	f0	4d
[Q]	...	15	...	15	f0	15
[R]	...	2d	...	2d	f0	2d
[S]	...	1b	...	1b	f0	1b
[T]	...	2c	...	2c	f0	2c
[U]	...	3c	...	3c	f0	3c
[V]	...	2a	...	2a	f0	2a
[W]	...	1d	...	1d	f0	1d
[X]	...	22	...	22	f0	22
[Y]	...	35	...	35	f0	35
[Z]	...	1a	...	1a	f0	1a

Some Keys' Scan Codes

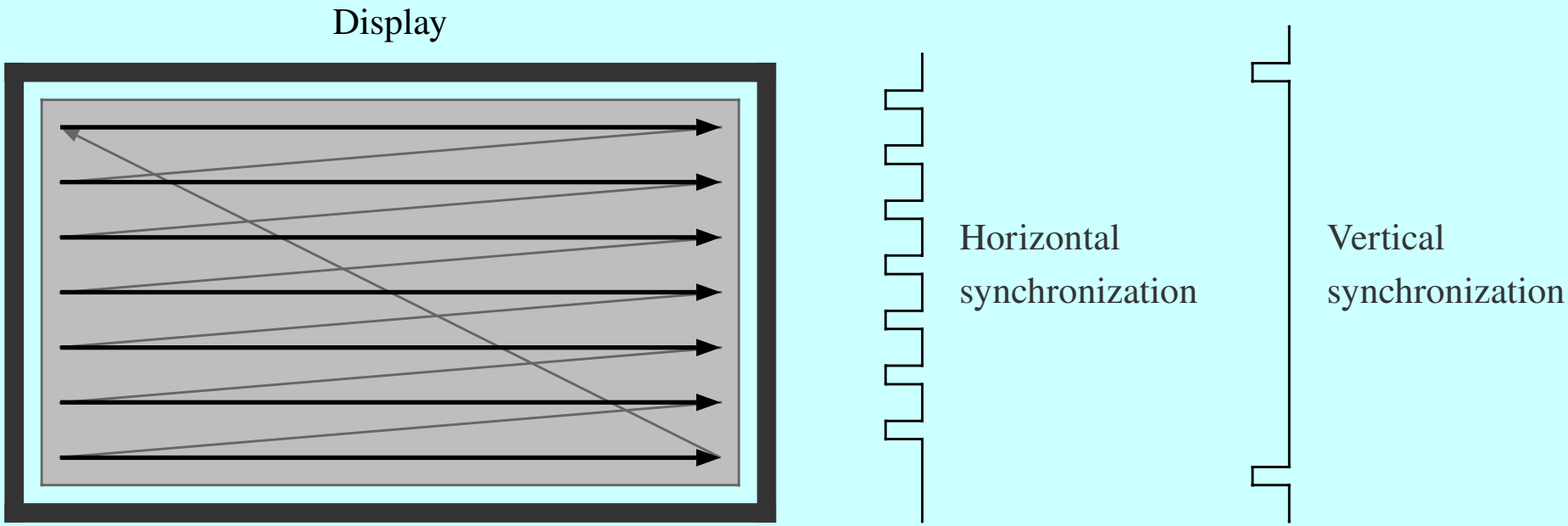
```
[- =] ...          4e ... 4e f0 4e
[^ ~] ...          55 ... 55 f0 55
[\ |] ...          6a ... 6a f0 6a
[@ `] ...          54 ... 54 f0 54
[[ {] ...          5b ... 5b f0 5b
[; +] ...          4c ... 4c f0 4c
[: *] ...          52 ... 52 f0 52
[] }] ...          5d ... 5d f0 5d
[, <] ...          41 ... 41 f0 41
[. >] ...          49 ... 49 f0 49
[/ ?] ...          4a ... 4a f0 4a
[\ _] ...          51 ... 51 f0 51

[Tab] ...          0d ... 0d f0 0d
[Enter] ...        5a ... 5a f0 5a
[Space] ...        29 ... 29 f0 29
[Back space] ...   66 ... 66 f0 66
[Esc] ...          76 ... 76 f0 76
[L Shift] ...      12 ... 12 f0 12
```

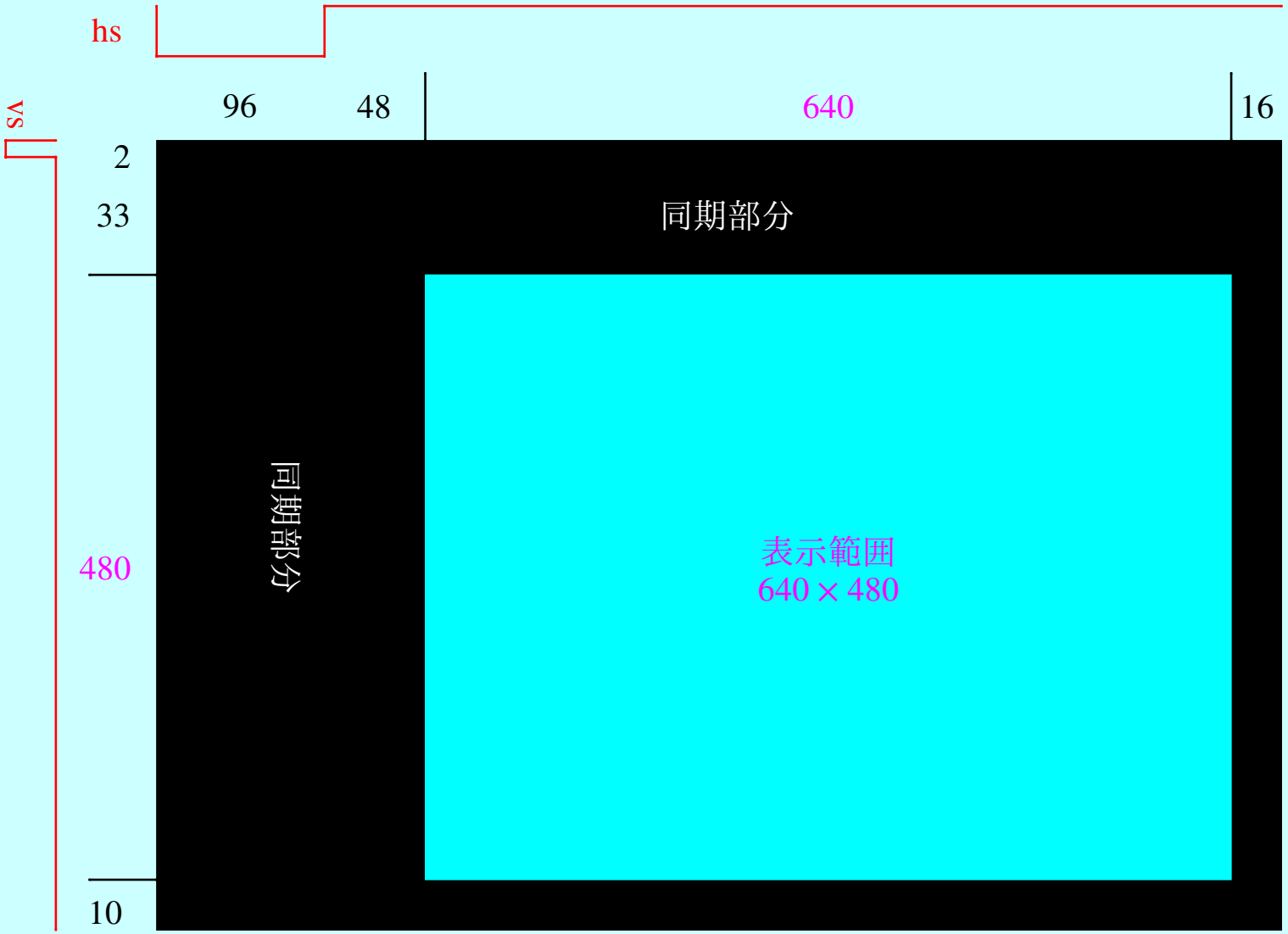
Some Keys' Scan Codes

[R Shift] ...	59 ... 59 f0 59
[L Ctrl] ...	14 ... 14 f0 14
[L Alt] ...	11 ... 11 f0 11
[R Ctrl] ...	e0 14 ... e0 14 e0 f0 14
[R Alt] ...	e0 11 ... e0 11 e0 f0 11
[L Arrow] ...	e0 6b ... e0 6b e0 f0 6b
[R Arrow] ...	e0 74 ... e0 74 e0 f0 74
[U Arrow] ...	e0 75 ... e0 75 e0 f0 75
[D Arrow] ...	e0 72 ... e0 72 e0 f0 72
[L Windows] ...	e0 1f ... e0 1f e0 f0 1f
[R Windows] ...	e0 27 ... e0 27 e0 f0 27
[Insert] ...	e0 70 ... e0 70 e0 f0 70
[Delete] ...	e0 71 ... e0 71 e0 f0 71
[Home] ...	e0 6c ... e0 6c e0 f0 6c
[End] ...	e0 69 ... e0 69 e0 f0 69
[PageUp] ...	e0 7d ... e0 7d e0 f0 7d
[PageDown] ...	e0 7a ... e0 7a e0 f0 7a
[Pause]	e1 14 77 e1 f0 14 f0 77

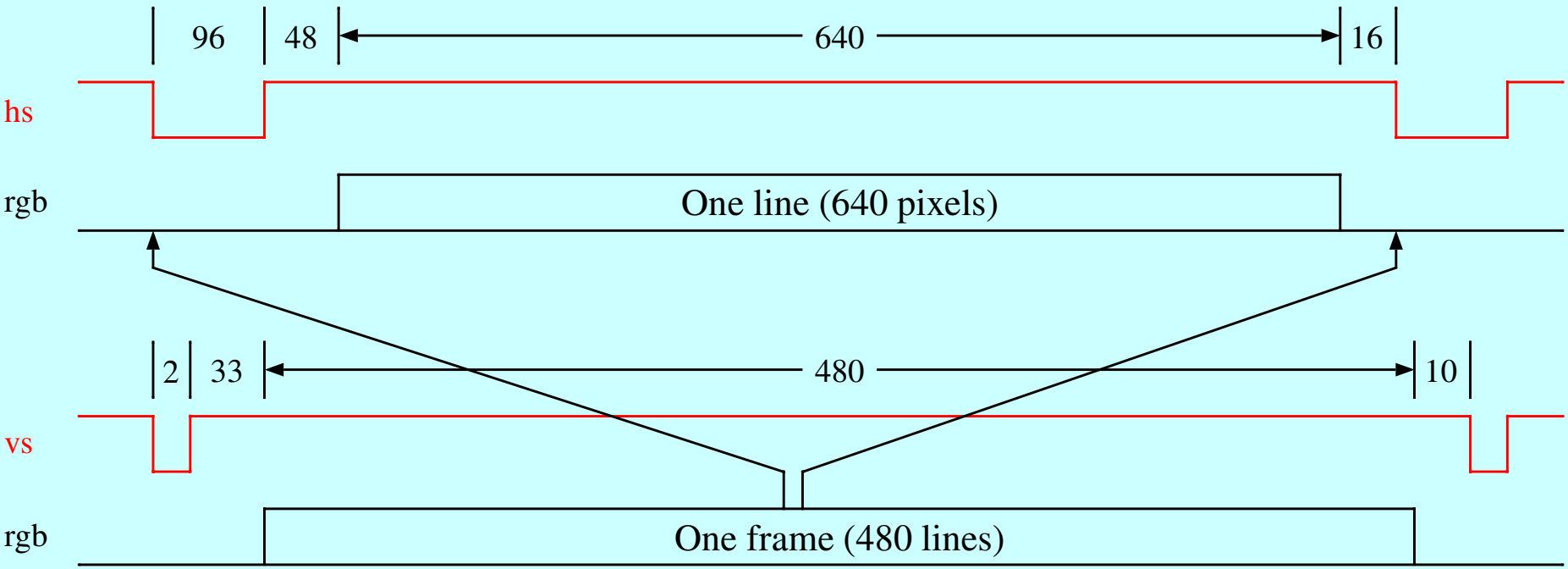
VGA: Video Graphics Array (Standard VGA: 640 × 480)



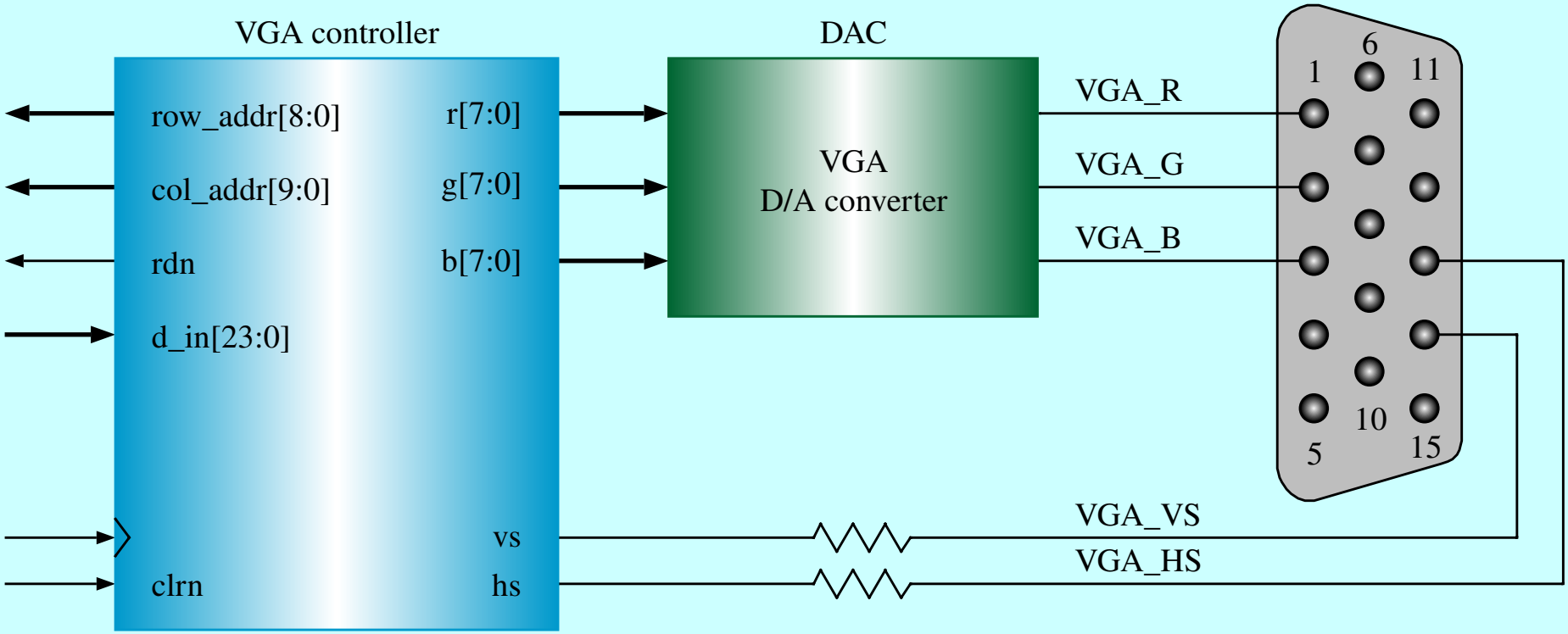
VGA Timing



VGA Timing


96
48
640
16
hs
rgb
One line (640 pixels)
2
33
480
10
vs
rgb
One frame (480 lines)

VGA Connector



RGB: 24 bits, true color (トゥルーカラー)

$2^{24} = 16{,}777{,}216$ 色（約 1677 万色）

vgac.v

```
‘timescale 1ns/1ns
module vgac (vga_clk,clrn,d_in,row_addr,col_addr,rdn,r,g,b,hs,vs);   // vgac
    input     [23:0] d_in;       // rrrrrrrr_gggggggg_bbbbbbbb, pixel
    input            vga_clk;    // 25MHz
    input            clrn;
    output reg [8:0] row_addr;   // pixel ram row address, 480 (512) lines
    output reg [9:0] col_addr;   // pixel ram col address, 640 (1024) pixels
    output reg [7:0] r,g,b;      // red, green, blue colors, 8-bit for each
    output reg       rdn;        // read pixel RAM (active low)
    output reg       hs,vs;      // horizontal and vertical synchronization
    // h_count: vga horizontal counter (0-799 pixels)
    reg [9:0] h_count;
    always @ (posedge vga_clk or negedge clrn) begin
        if (!clrn) begin
            h_count <= 10’h0;
        end else if (h_count == 10’d799) begin
            h_count <= 10’h0;
        end else begin
            h_count <= h_count + 10’h1;
        end
    end
```

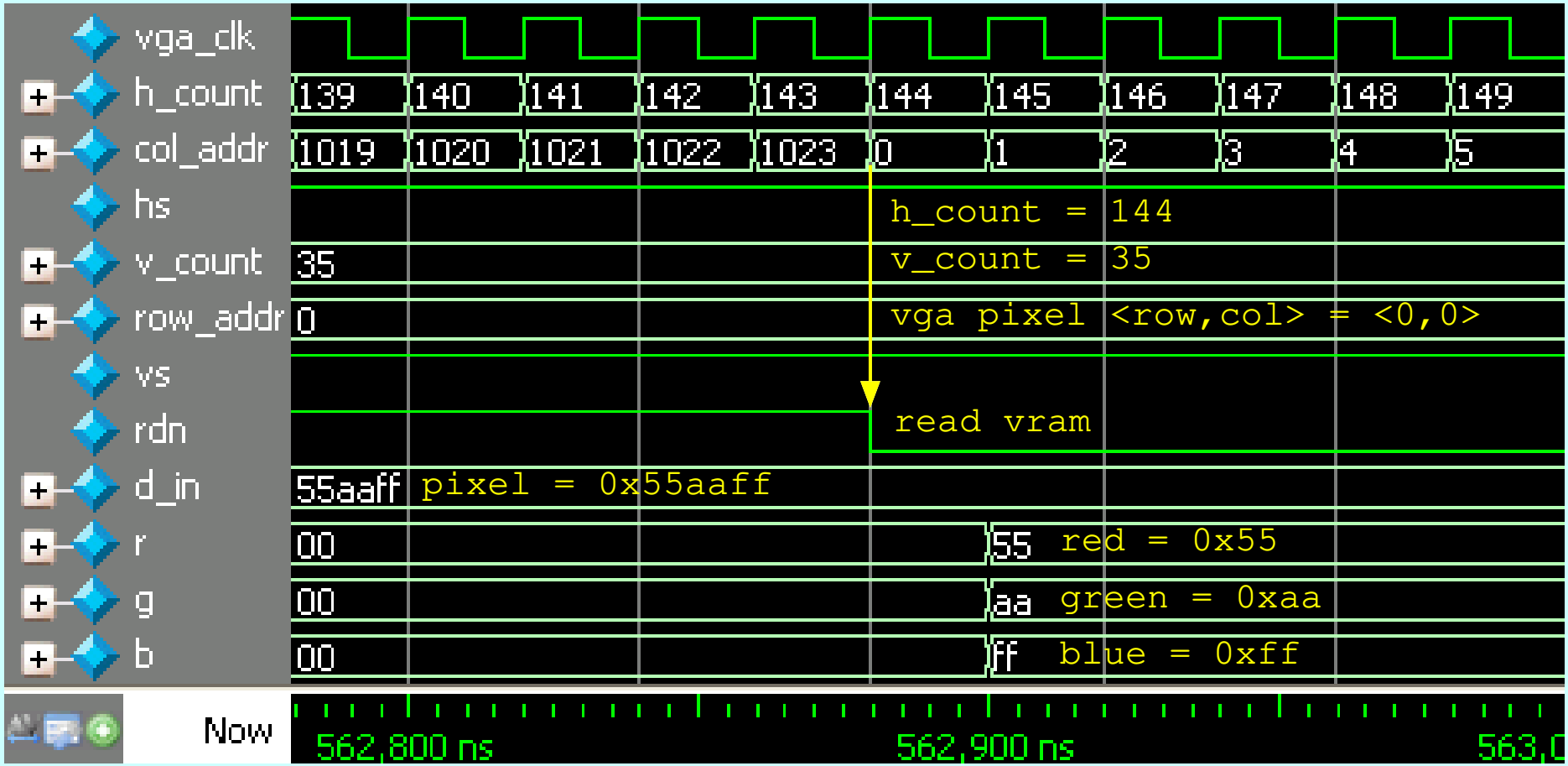
vgac.v

```
// v_count: vga vertical counter (0-524 lines)
reg [9:0] v_count;
always @ (posedge vga_clk or negedge clrn) begin
    if (!clrn) begin
        v_count <= 10'h0;
    end else if (h_count == 10'd799) begin
        if (v_count == 10'd524) begin
            v_count <= 10'h0;
        end else begin
            v_count <= v_count + 10'h1;
        end
    end
end
```

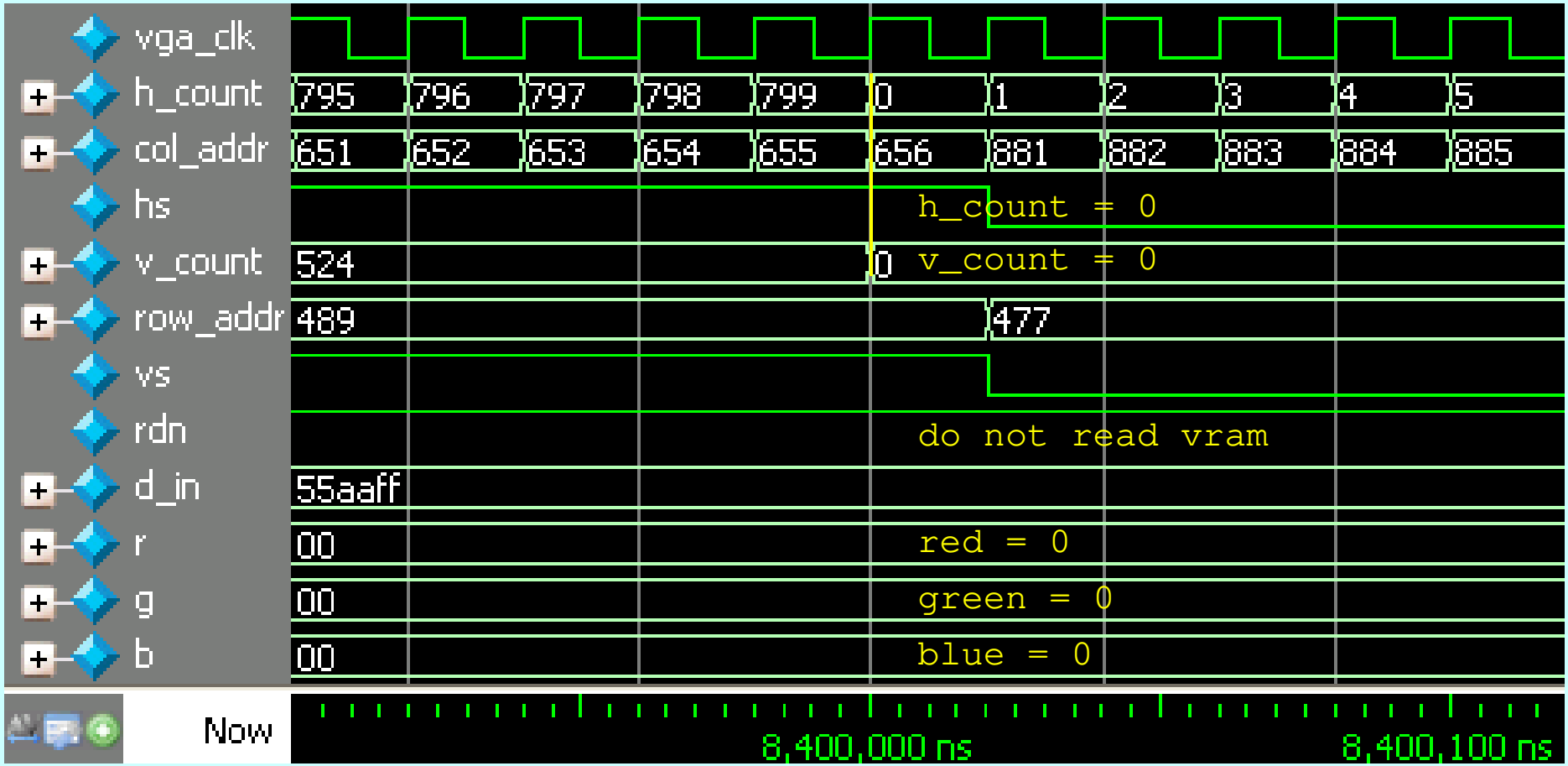
vgac.v

```
    // signals, will be latched for outputs
    wire  [9:0] row    =  v_count - 10'd35;       // pixel ram row address
    wire  [9:0] col    =  h_count - 10'd143;      // pixel ram col address
    wire        h_sync = (h_count > 10'd95);      //  96 -> 799
    wire        v_sync = (v_count > 10'd1);       //   2 -> 524
    wire        read   = (h_count > 10'd142) &&   // 143 -> 782 =
                         (h_count < 10'd783) &&   //               640 pixels
                         (v_count > 10'd34)  &&   //  35 -> 514 =
                         (v_count < 10'd515);     //               480 lines
    // vga signals
    always @ (posedge vga_clk) begin
        row_addr <=  row[8:0];                    // pixel ram row address
        col_addr <=  col;                         // pixel ram col address
        rdn      <= ~read;                        // read pixel (active low)
        hs       <=  h_sync;                      // horizontal synch
        vs       <=  v_sync;                      // vertical   synch
        r        <=  rdn ? 8'h0 : d_in[23:16];    // 8-bit red
        g        <=  rdn ? 8'h0 : d_in[15:08];    // 8-bit green
        b        <=  rdn ? 8'h0 : d_in[07:00];    // 8-bit blue
    end
endmodule
```

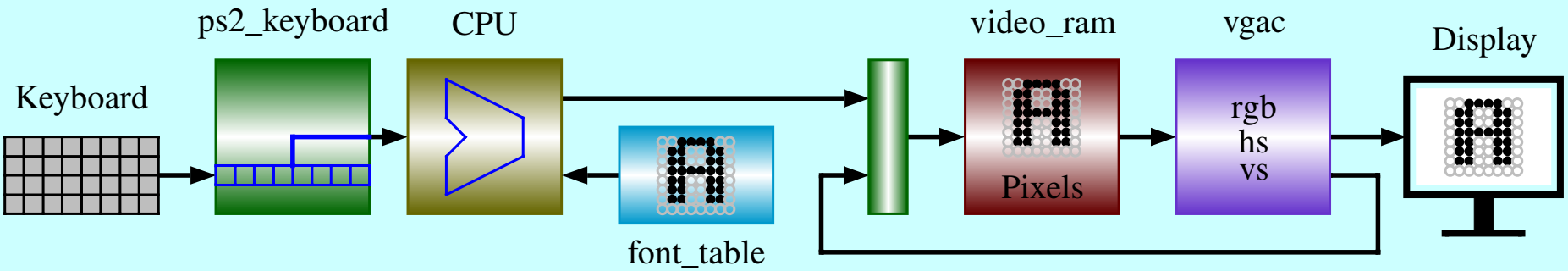
Simulation (First Pixel)



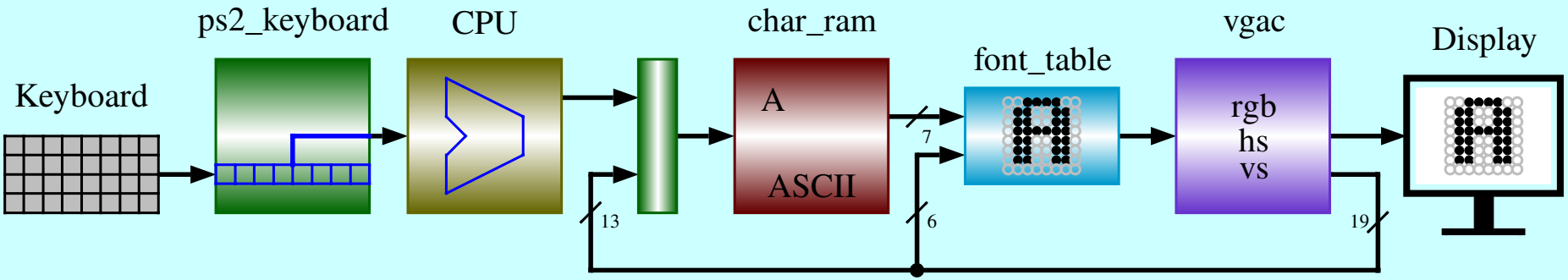
Simulation (One Frame)



Graphics Mode and Text Mode of VGA

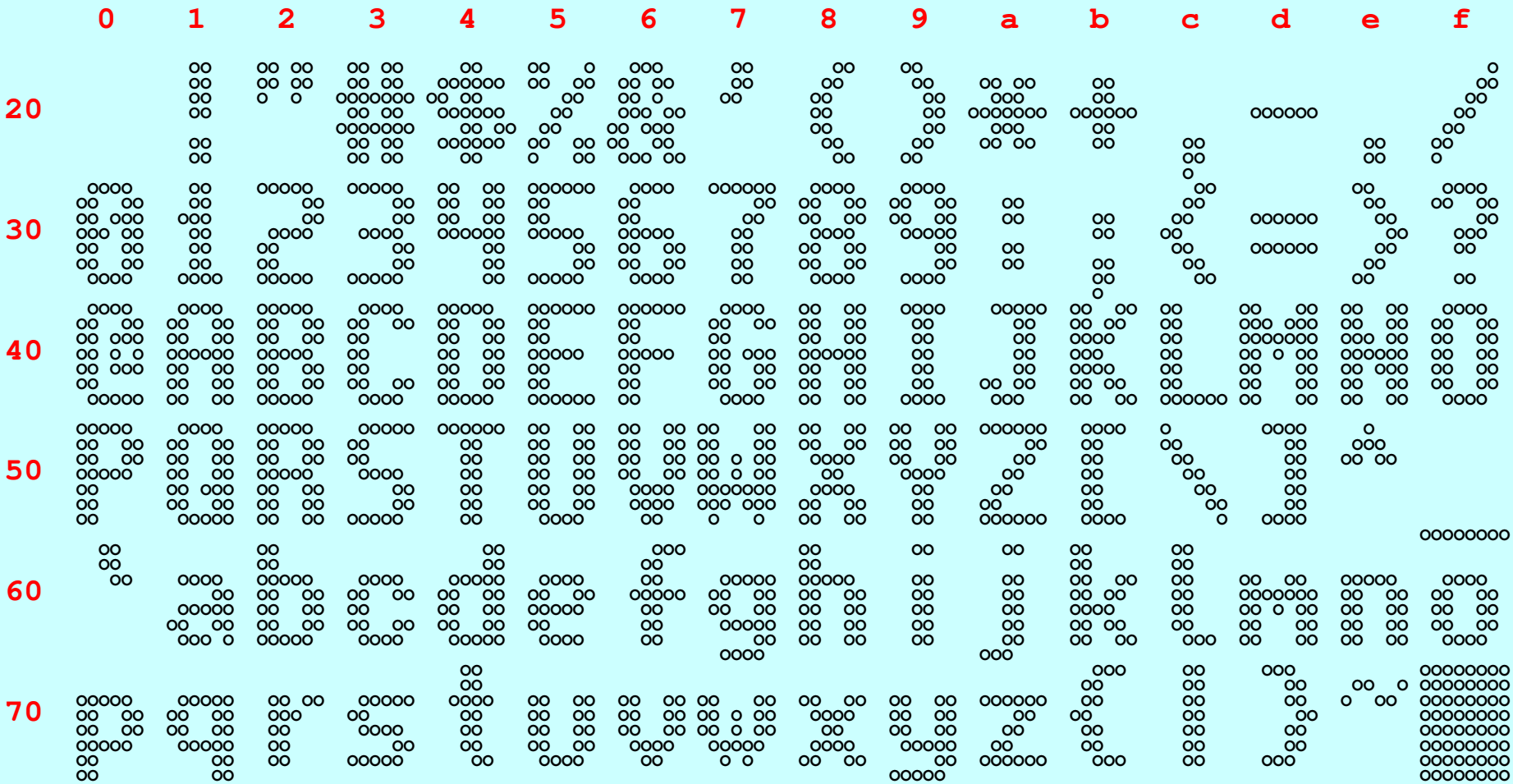


(a) Graphics mode (store pixels to RAM)

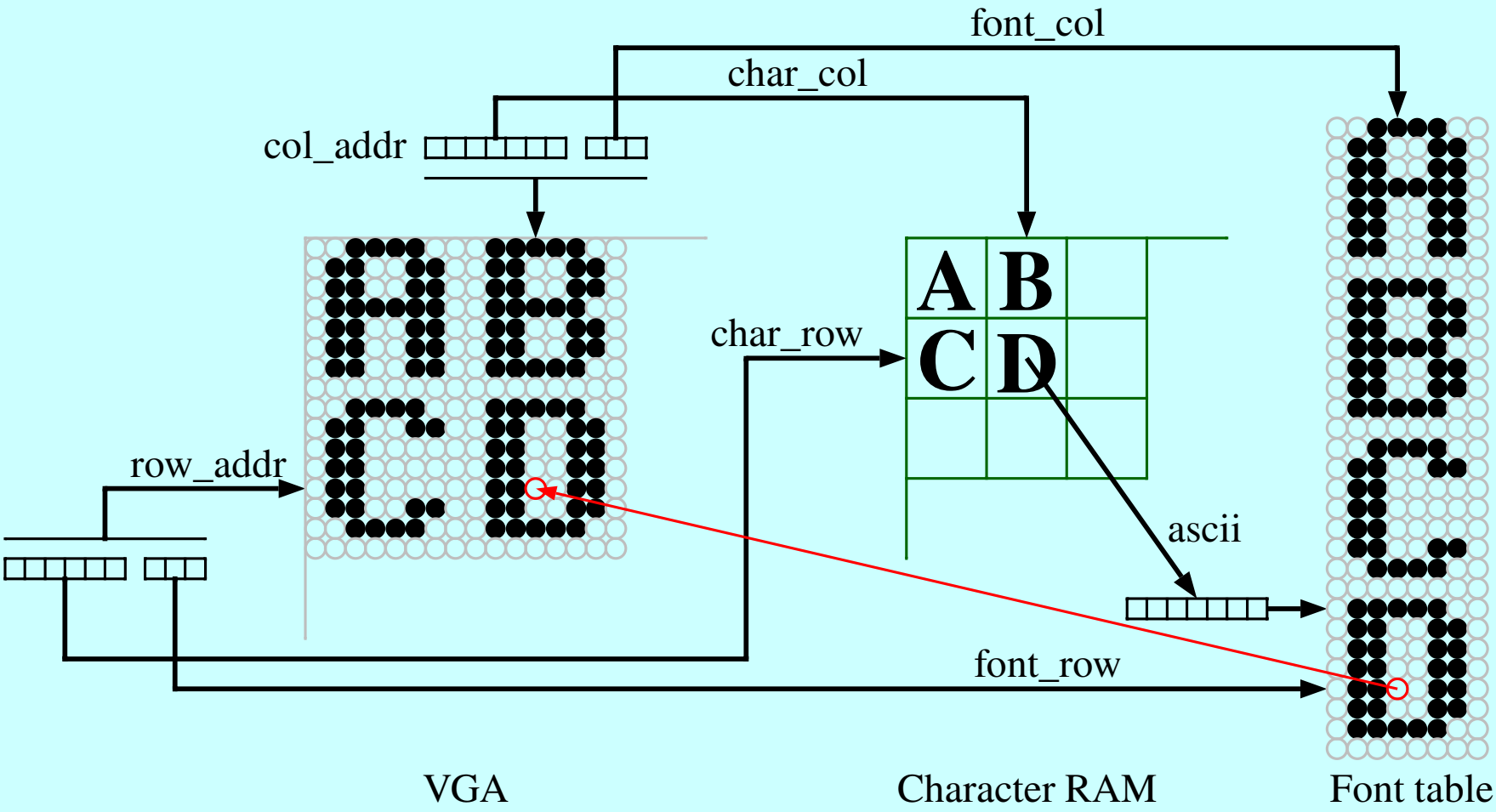


(b) Text mode (store ASCII codes to RAM)

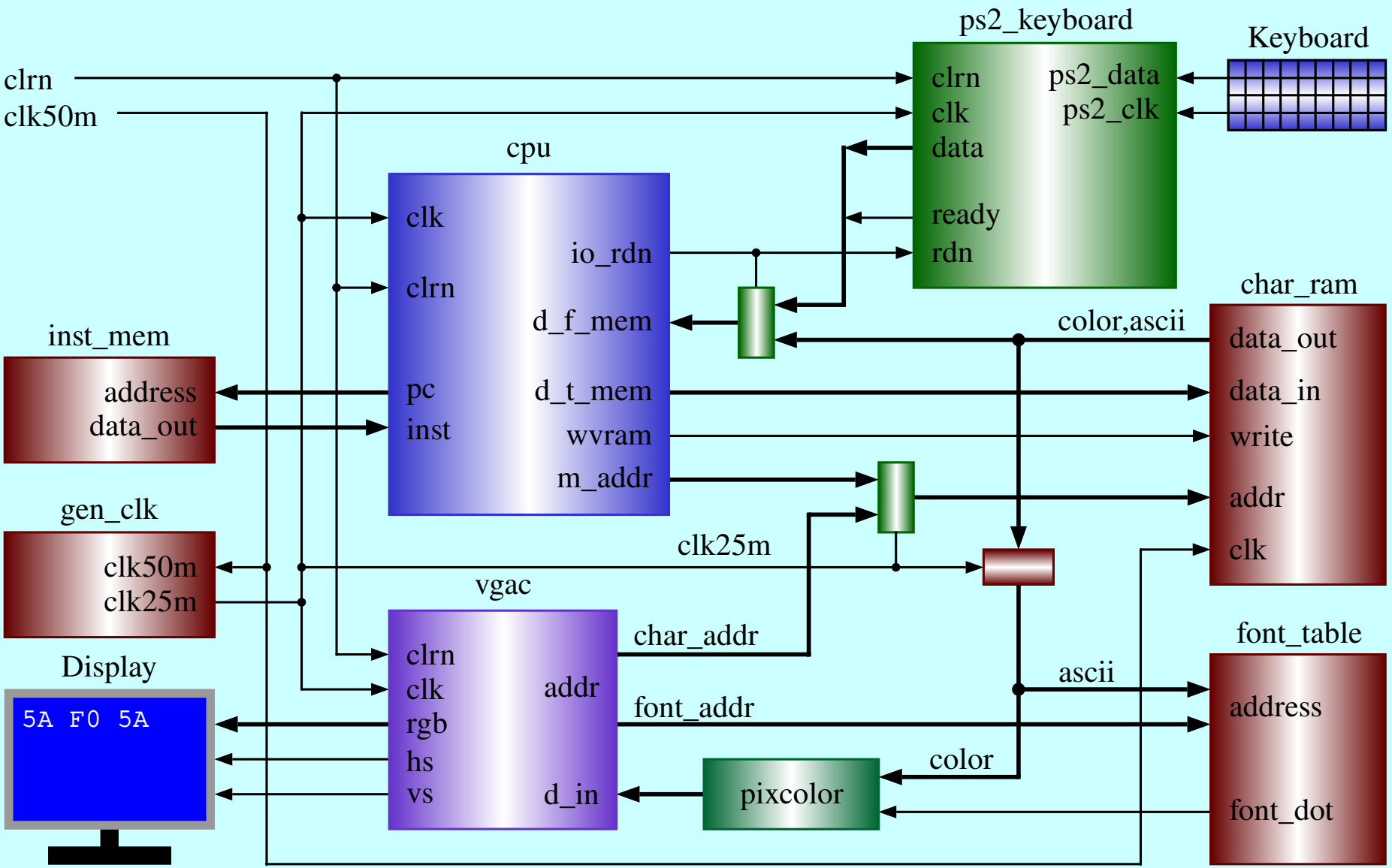
ASCII Font Example



Character Address and Font Address



CPU + キーボード + ディスプレイ



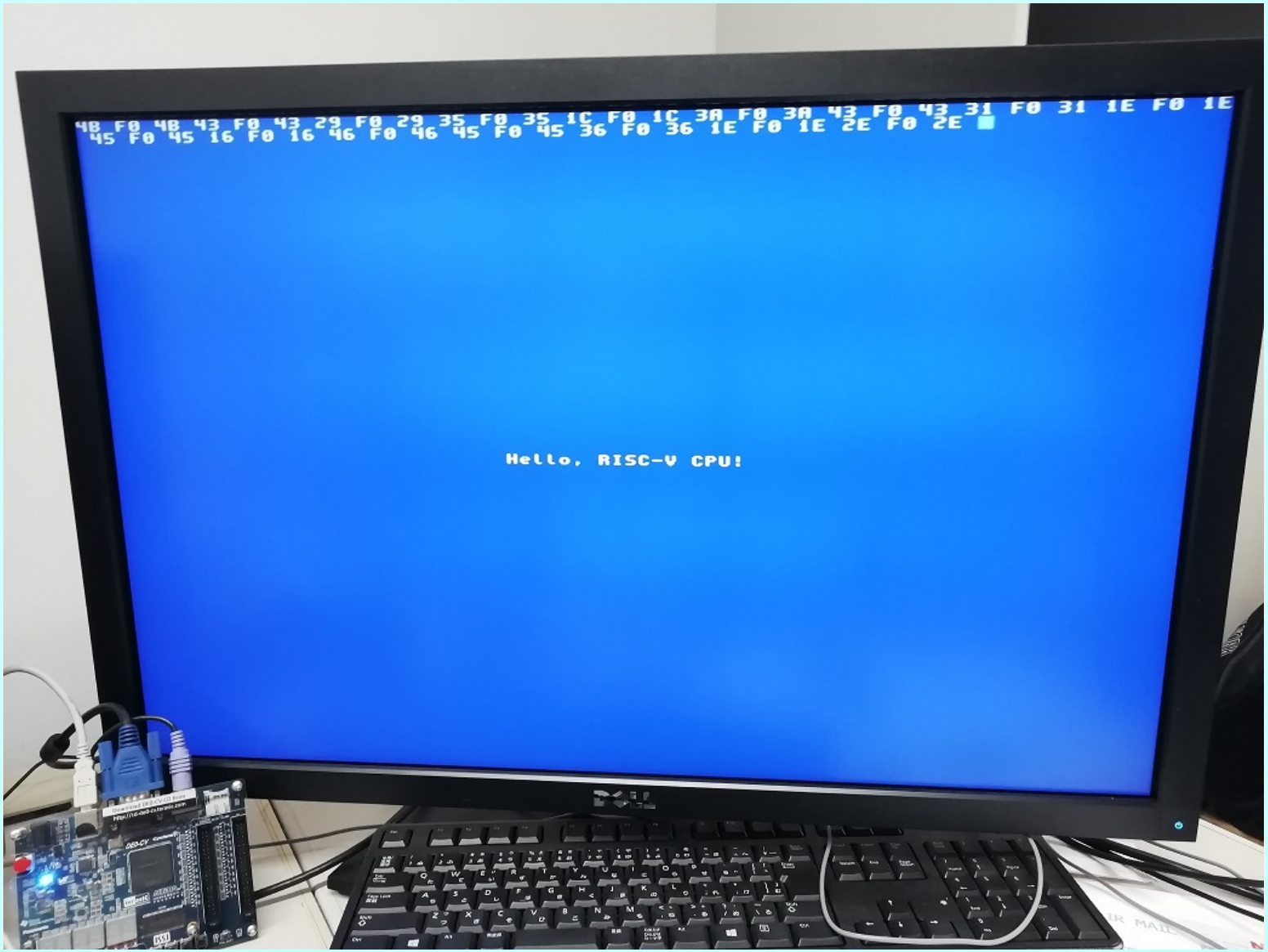
display_scan_codes.s

```
.text                       # code segment
main:
    lui   s2,  0xc0000      # vram space: c0000000 - dfffffff
read_kbd:
    csrr  a0,  0x800        # read kbd: {0,ready,byte}
    andi  a1,  a0,  0x100   # check if ready
    beqz  a1,  read_kbd     # if no key pressed, wait
    andi  a1,  a0,  0xff    # ready, get data
    srli  a0,  a1,  4       # first digit
    call  to_ascii
    call  display
    andi  a0,  a1,  0xf     # second digit
    call  to_ascii
    call  display
    li    a0,  0x20         # third [Space]
```

display_scan_codes.s

```
    call  display
    j     read_kbd          # check next
to_ascii:
    addi  t2,  a0,  -9
    bgtz  t2,  abcdef
    addi  a0,  a0,  0x30    # to ascii [0-9]
    ret
abcdef:
    addi  a0,  a0,  0x37    # to ascii [a-f]
    ret                     # a0: ASCII
display:                    # a0: ASCII
    sw    a0,  0(s2)        # to vram
    addi  s2,  s2,  4       # vram address++
    ret                     # a0: void
.end
```

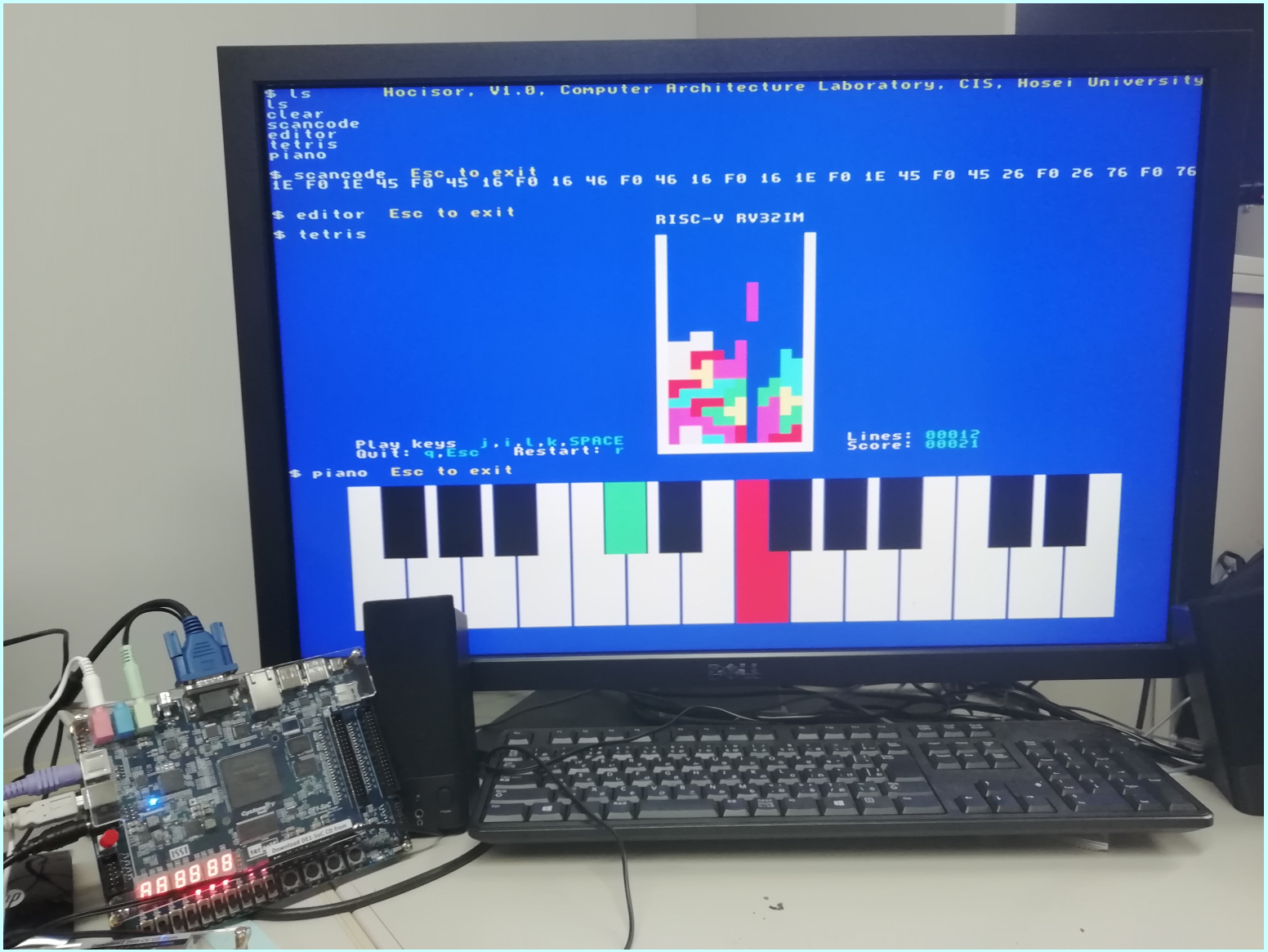
Display Scan Codes



Key's Scan Code Examples

```
[Tab]...          0d ... 0d f0 0d // 0d: Make Code; f0 0d: Break Code
[Enter]...        5a ... 5a f0 5a // ...: held down without being released
[Space]...        29 ... 29 f0 29
[Back space]...   66 ... 66 f0 66
[Esc]...          76 ... 76 f0 76
[L Shift]...      12 ... 12 f0 12
[R Shift]...      59 ... 59 f0 59
[L Ctrl]...       14 ... 14 f0 14
[L Alt]...        11 ... 11 f0 11
[R Ctrl]...       e0 14 ... e0 14 e0 f0 14     // Extended Keys (plus e0)
[R Alt]...        e0 11 ... e0 11 e0 f0 11
[L Arrow]...      e0 6b ... e0 6b e0 f0 6b
[R Arrow]...      e0 74 ... e0 74 e0 f0 74
[U Arrow]...      e0 75 ... e0 75 e0 f0 75
[D Arrow]...      e0 72 ... e0 72 e0 f0 72
[L Windows]...    e0 1f ... e0 1f e0 f0 1f
[R Windows]...    e0 27 ... e0 27 e0 f0 27
[Insert]...       e0 70 ... e0 70 e0 f0 70
[Delete]...       e0 71 ... e0 71 e0 f0 71
[Home]...         e0 6c ... e0 6c e0 f0 6c
[End]...          e0 69 ... e0 69 e0 f0 69
[PageUp]...       e0 7d ... e0 7d e0 f0 7d
[PageDown]...     e0 7a ... e0 7a e0 f0 7a
[Pause]           e1 14 77 e1 f0 14 f0 77     // no break code, no repeat
```

CPU + キーボード + ディスプレイ



課題 XII (100 点 + 100 点)

1. The Operating System may check to see whether multiple keys are being pressed at the same time. Write all possible sequences of scancodes received by the keyboard controller when three keys ‘Ctrl’, ‘Alt’, and ‘Delete’ are pressed at the same time without releasing. Note that the ‘Ctrl’ and ‘Alt’ keys are pressed firstly in any order, and the ‘Delete’ key is pressed lastly. Also note that there are two ‘Ctrl’ keys and two ‘Alt’ keys. Among them, which sequences do the function in your PC?

2. Explain the interrupt and DMA and why the interrupt and DMA are required in modern computers.

課題 XII (100 点 + 100 点)

3. Option（+100 点）: Design an RISC-V computer system that can show the keyboard's scan codes on a display monitor (see P46 and refer to https://yamin.cis.k.hosei.ac.jp/rivasm/).