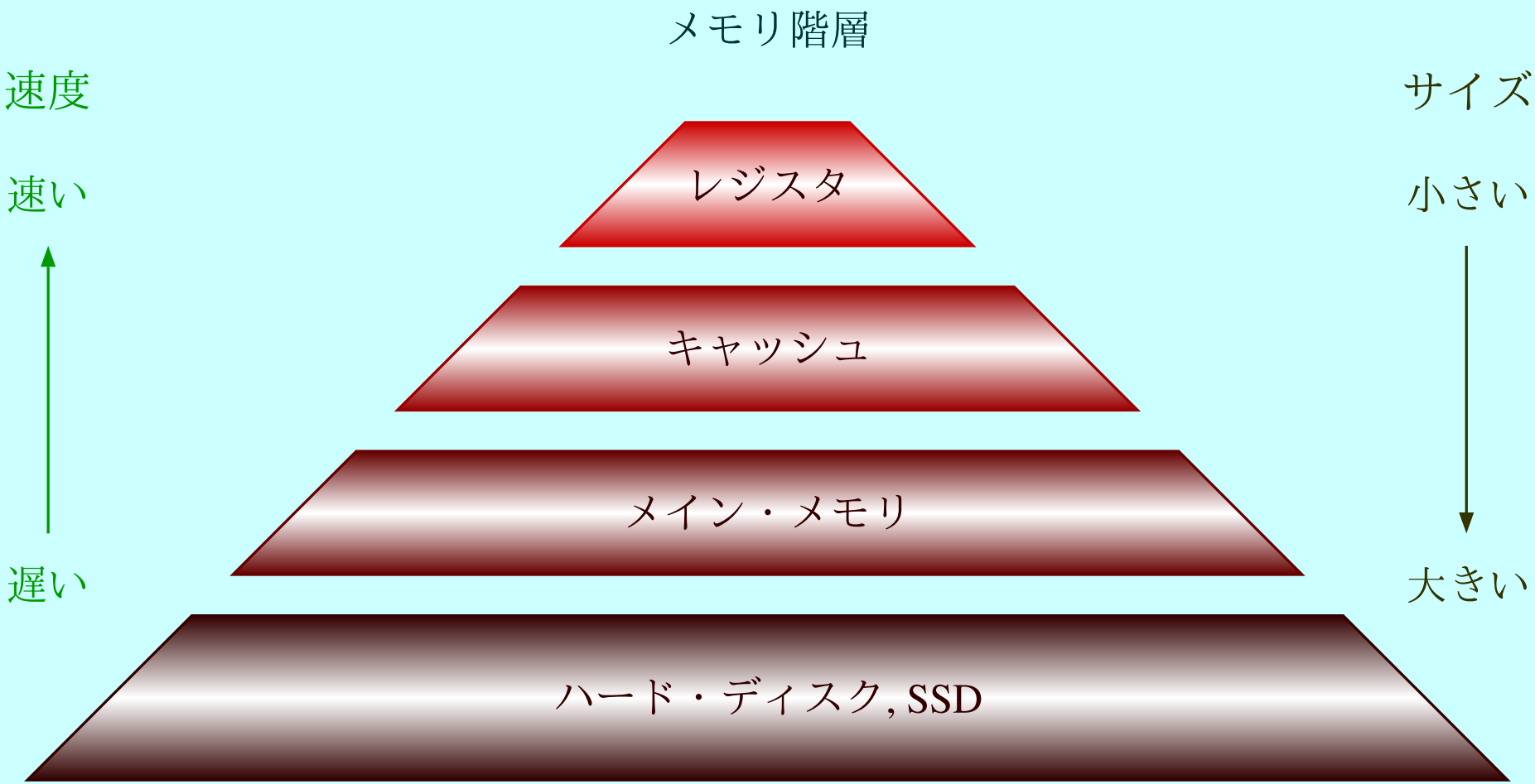
コンピュータ構成と設計（11）

メモリ階層

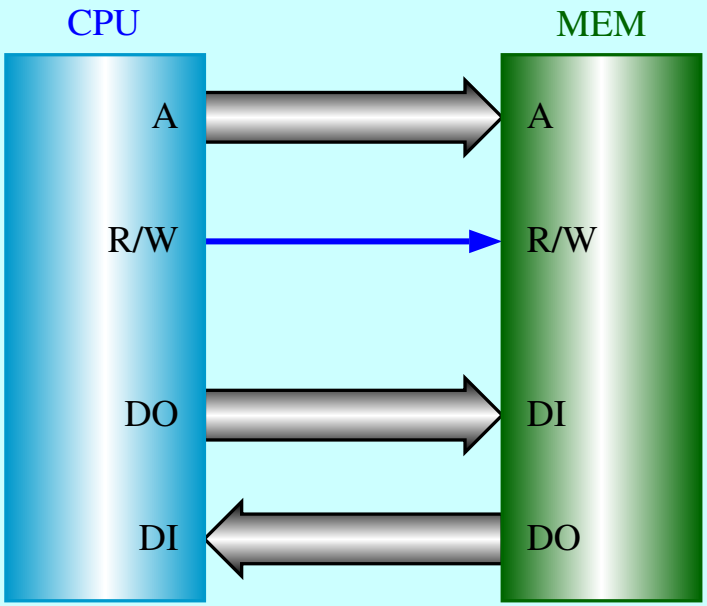
李　亜民

2024 年 12 月 12 日 (木)

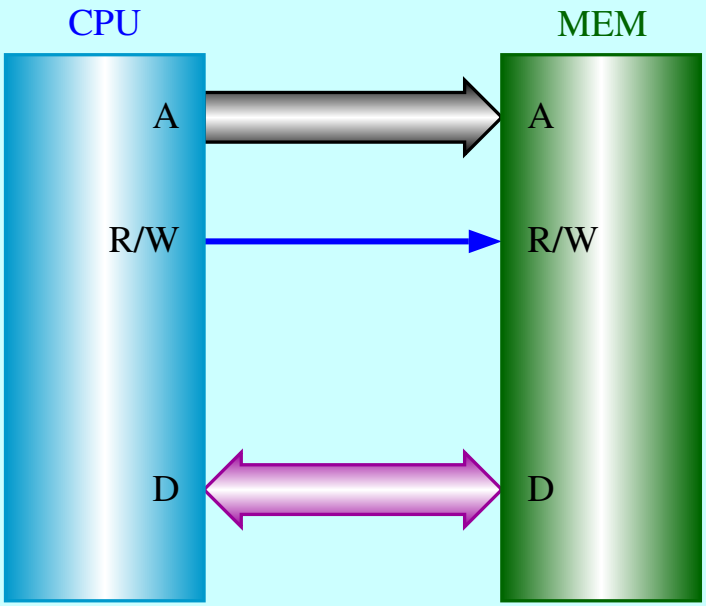
メモリ階層



メモリとCPU



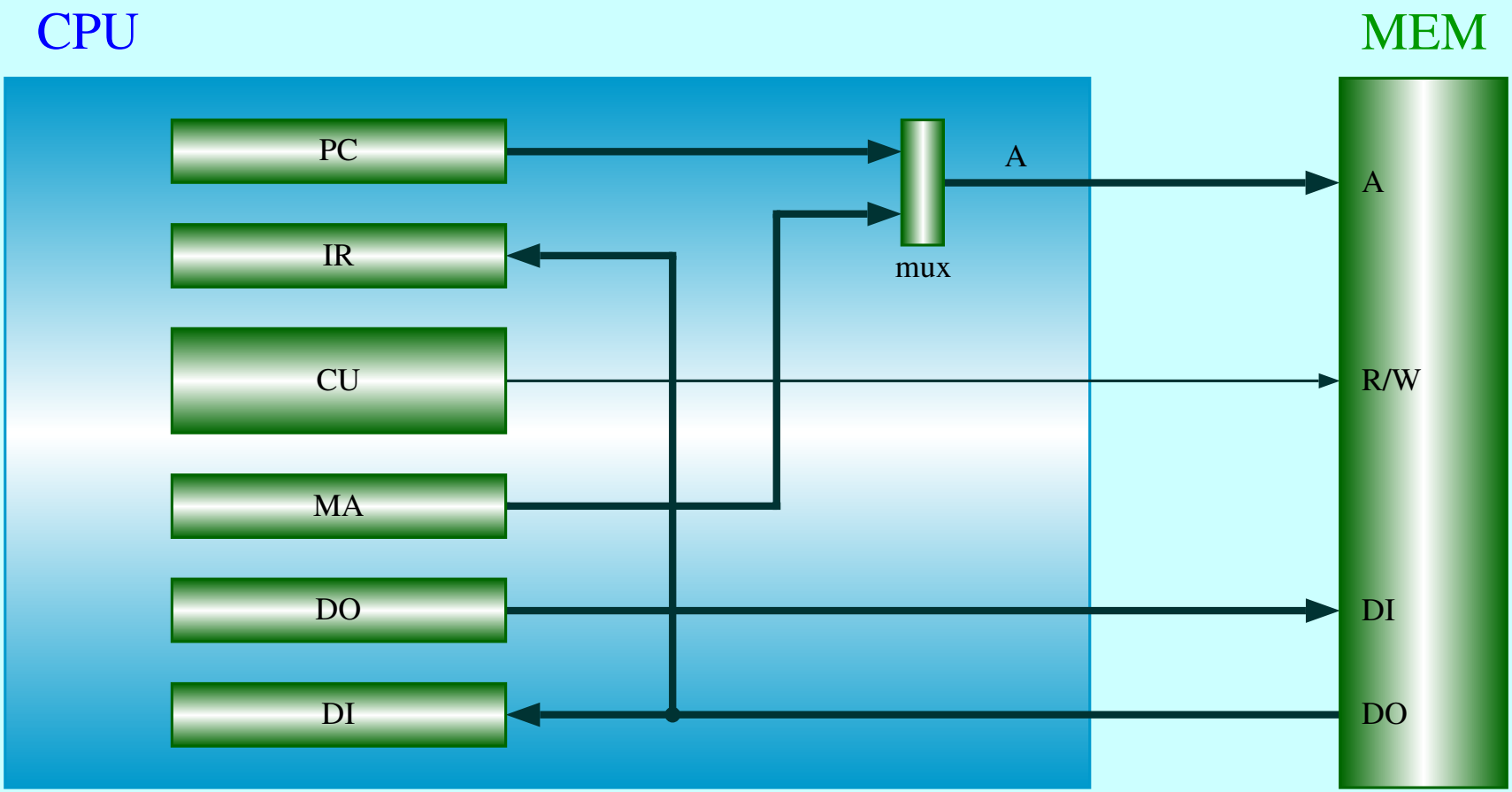
(a) 独立の単方向データバス



(b) 単一の双方向データバス

A (Address) : アドレスバス
D (Data) : データバス
DI (Data In) : データバス
DO (Data Out) : データバス
R/W (Read/Write) : リード/ライト信号

メモリとCPU



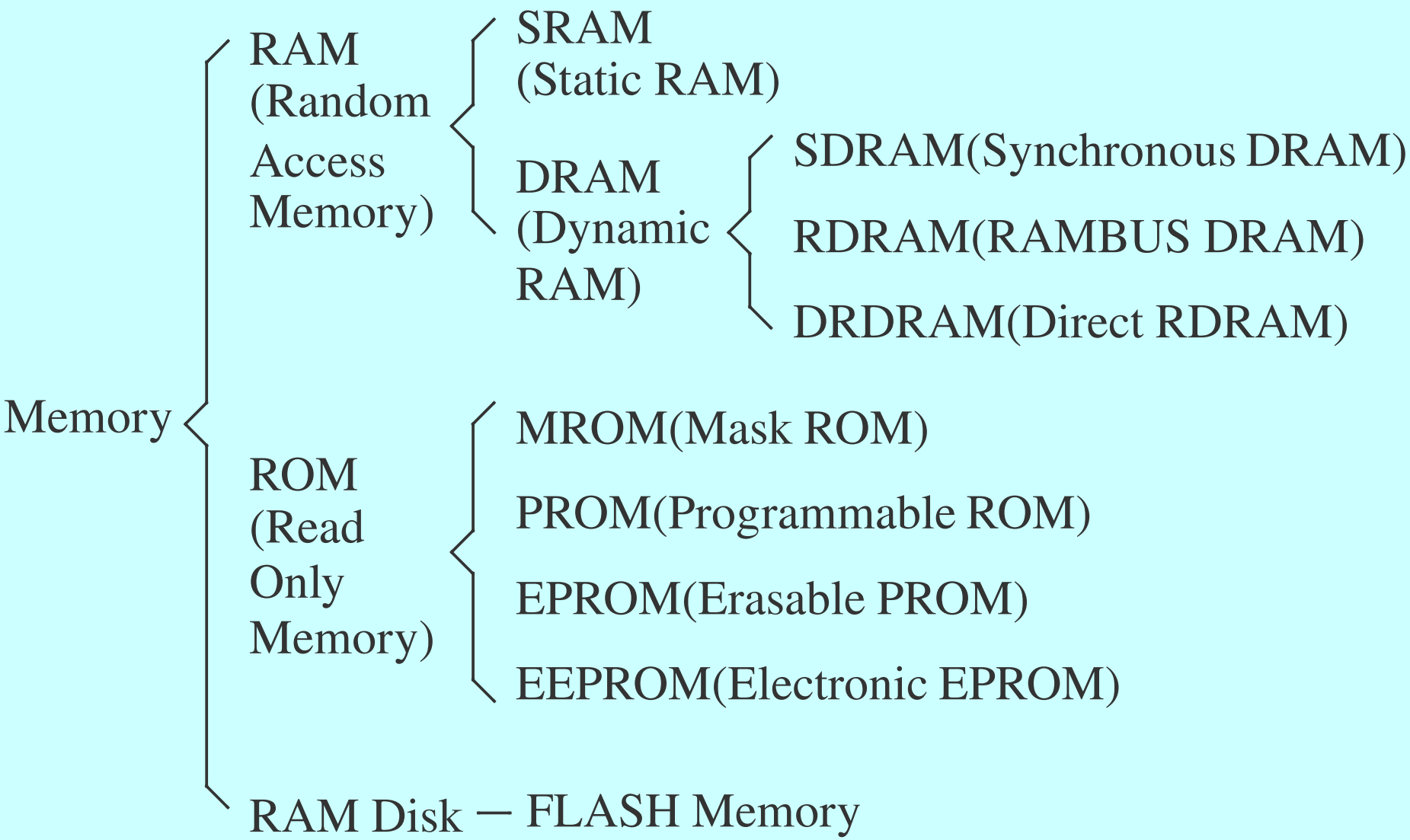
MA (Memory Address) : メモリアドレス
IR (Instruction Register) : 命令レジスタ

メモリ容量

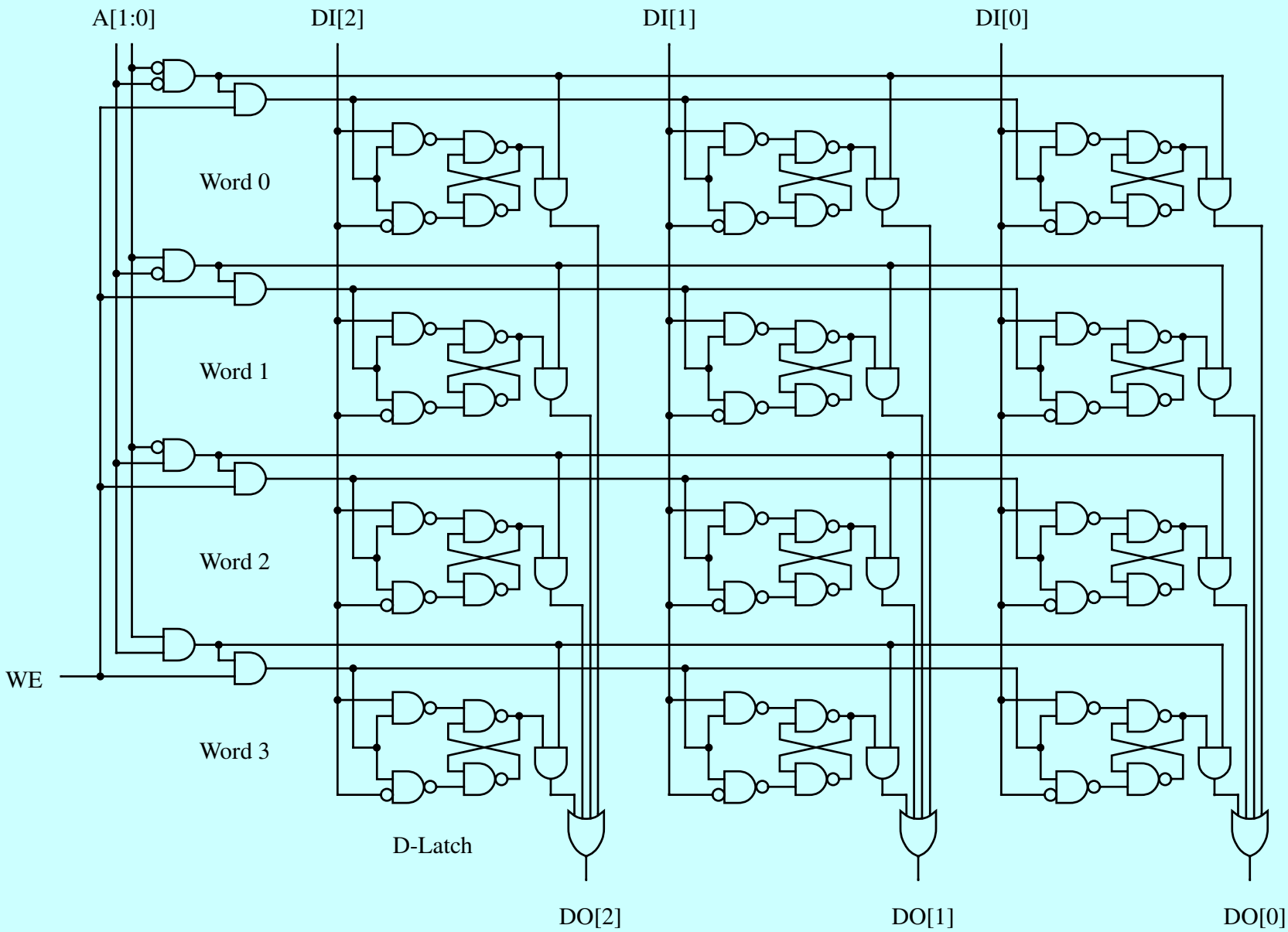
- メモリ容量はバイトで表される。1 バイトは 8 ビットに等しい
- 例えば，“メイン・メモリの容量は 8 ギガ・バイトである” という感じに使う

単位	10 進法での意味	2 進法での意味
K（キロ）	10^3	$2^{10} = 1\,024$
M（メガ）	10^6	$2^{20} = 1\,048\,576$
G（ギガ）	10^9	$2^{30} = 1\,073\,741\,824$
T（テラ）	10^{12}	$2^{40} = 1\,099\,511\,627\,776$

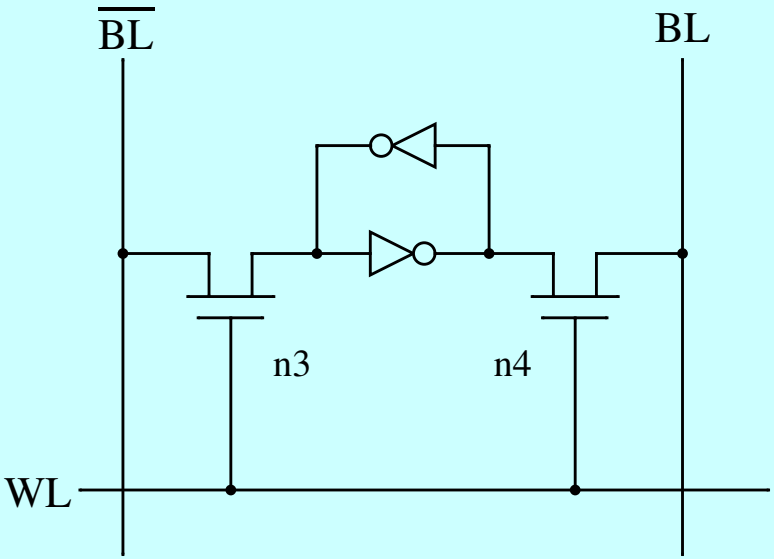
メモリの種類


Memory
RAM (Random Access Memory)
SRAM (Static RAM)
DRAM (Dynamic RAM)
SDRAM(Synchronous DRAM)
RDRAM(RAMBUS DRAM)
DRDRAM(Direct RDRAM)
ROM (Read Only Memory)
MROM(Mask ROM)
PROM(Programmable ROM)
EPROM(Erasable PROM)
EEPROM(Electronic EPROM)
RAM Disk ー FLASH Memory

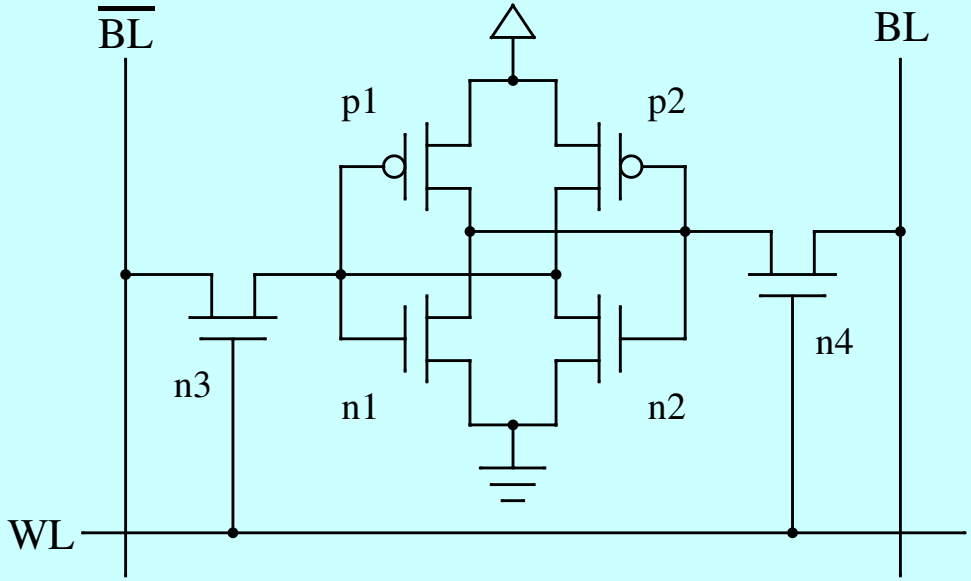
SRAM – 4 Words * 3 Bits/Word



SRAMセル回路

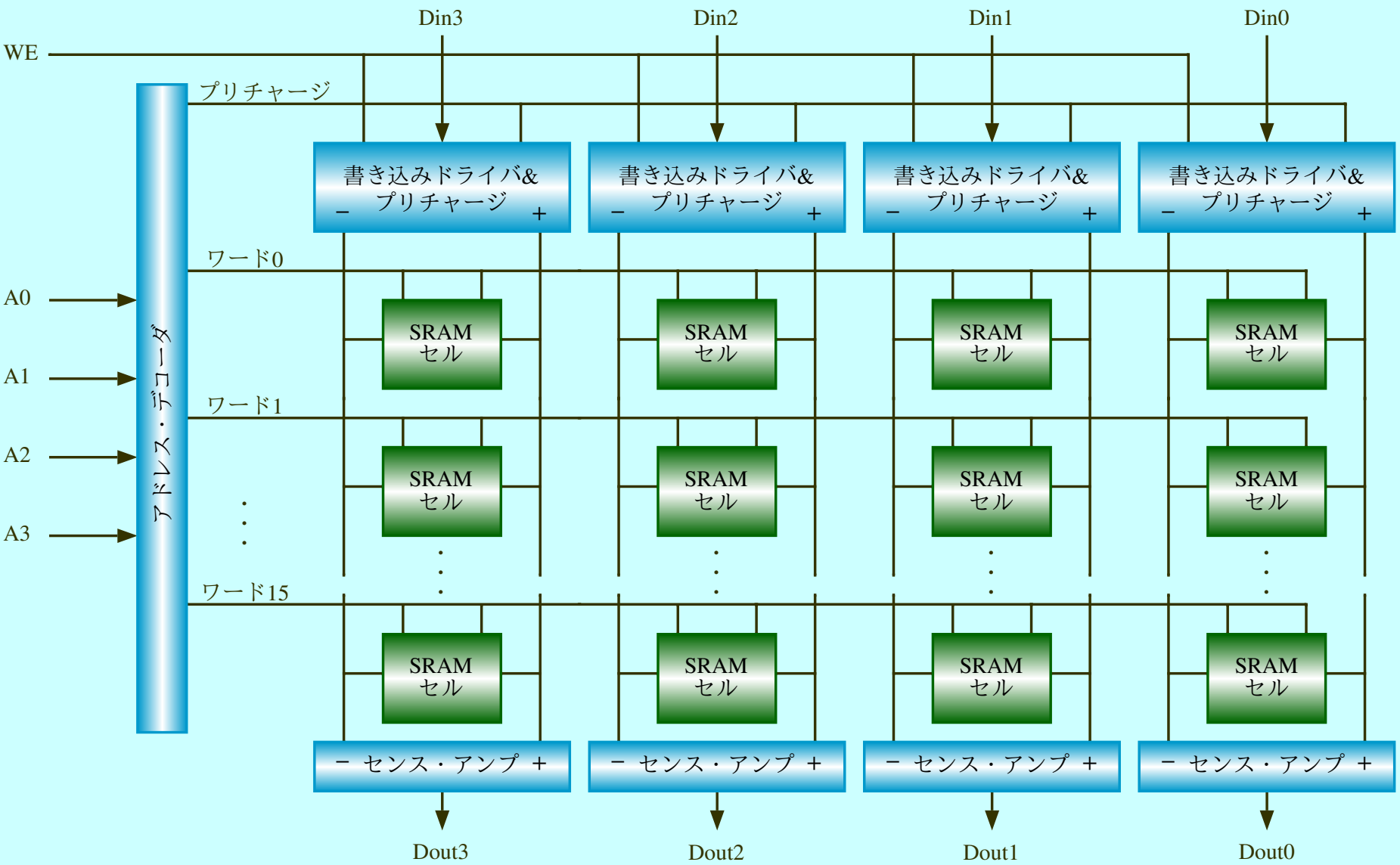


(a) RAMセル

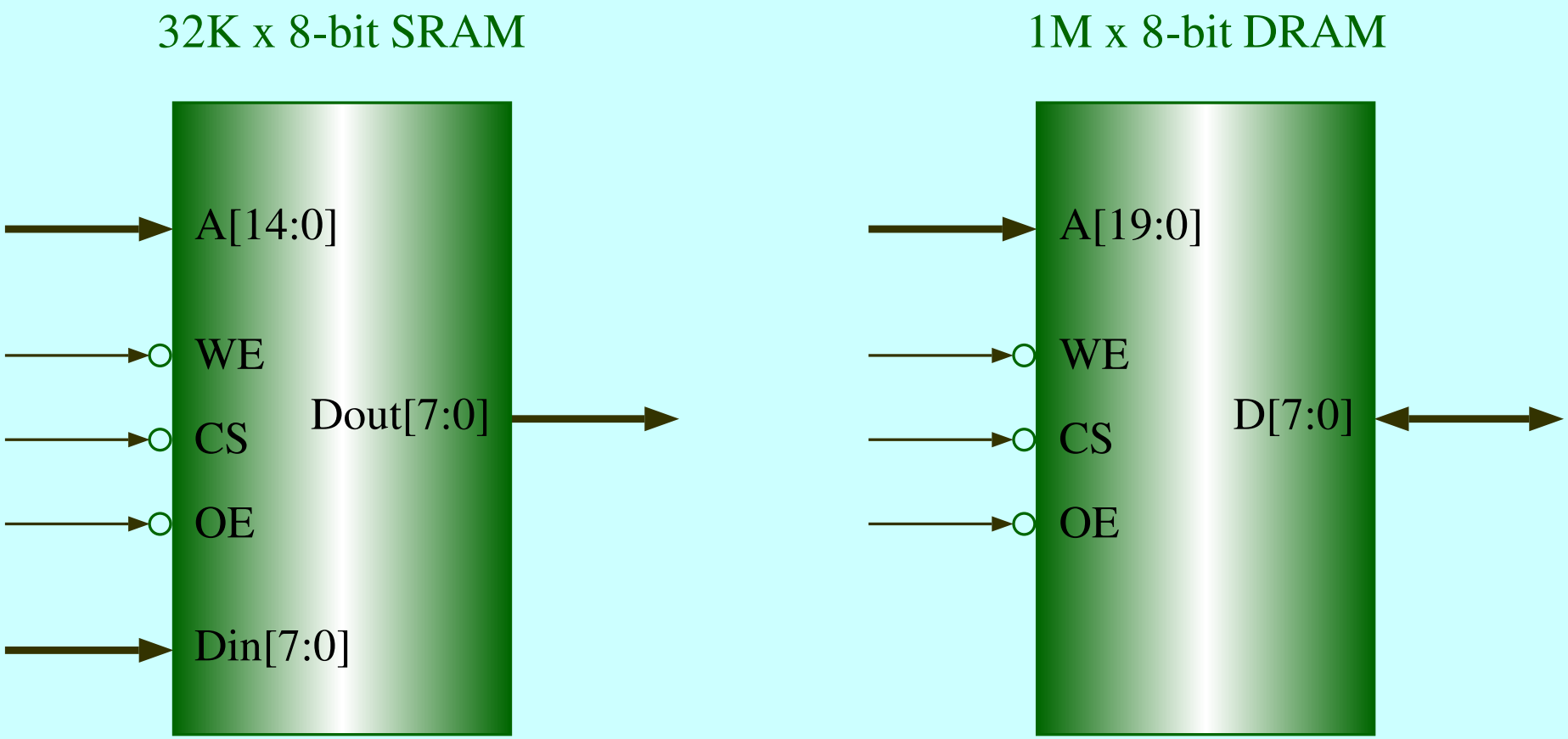


(b) RAMセル回路

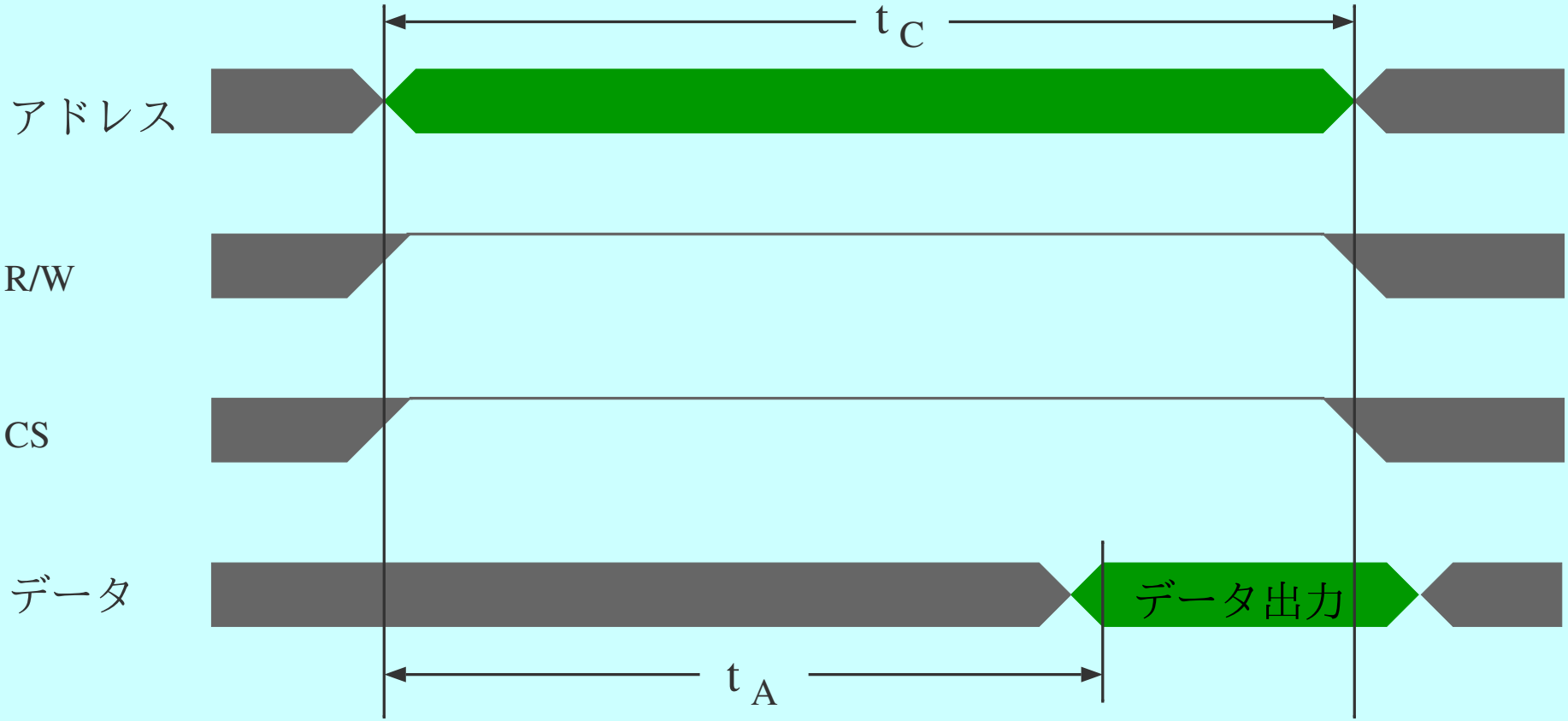
SRAM: 16 Words * 4 bits



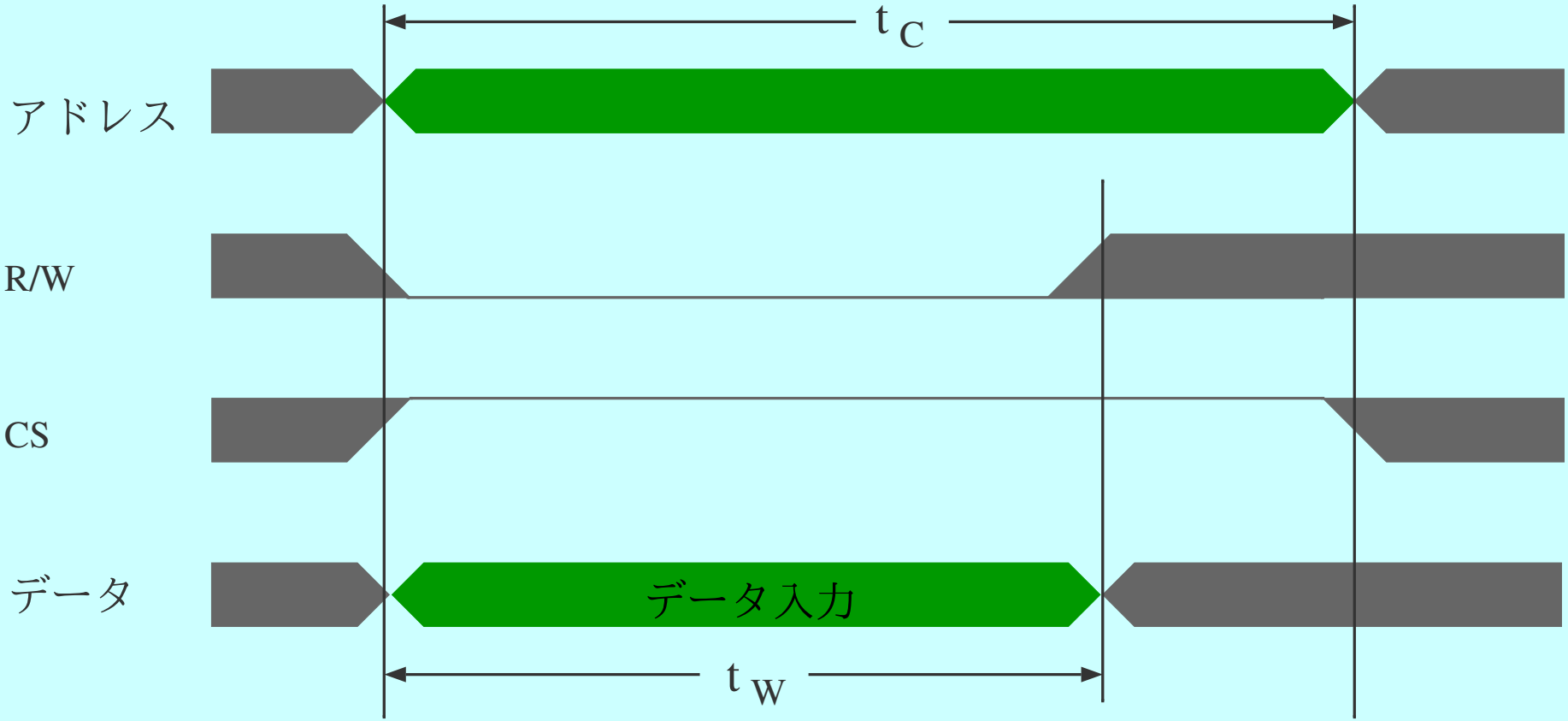
SRAMチップの例


32K x 8-bit SRAM
A[14:0]
WE
CS
OE
Dout[7:0]
Din[7:0]
1M x 8-bit DRAM
A[19:0]
WE
CS
OE
D[7:0]

SRAM — 読み出し

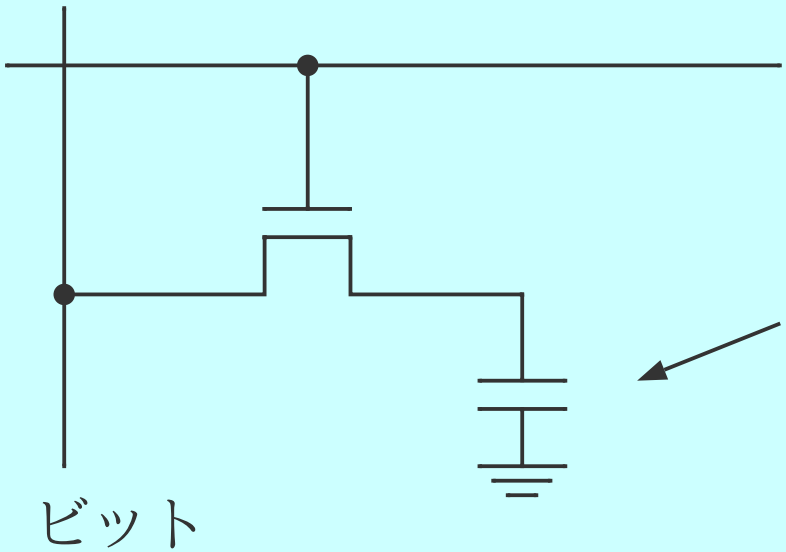


SRAM — 書き込み



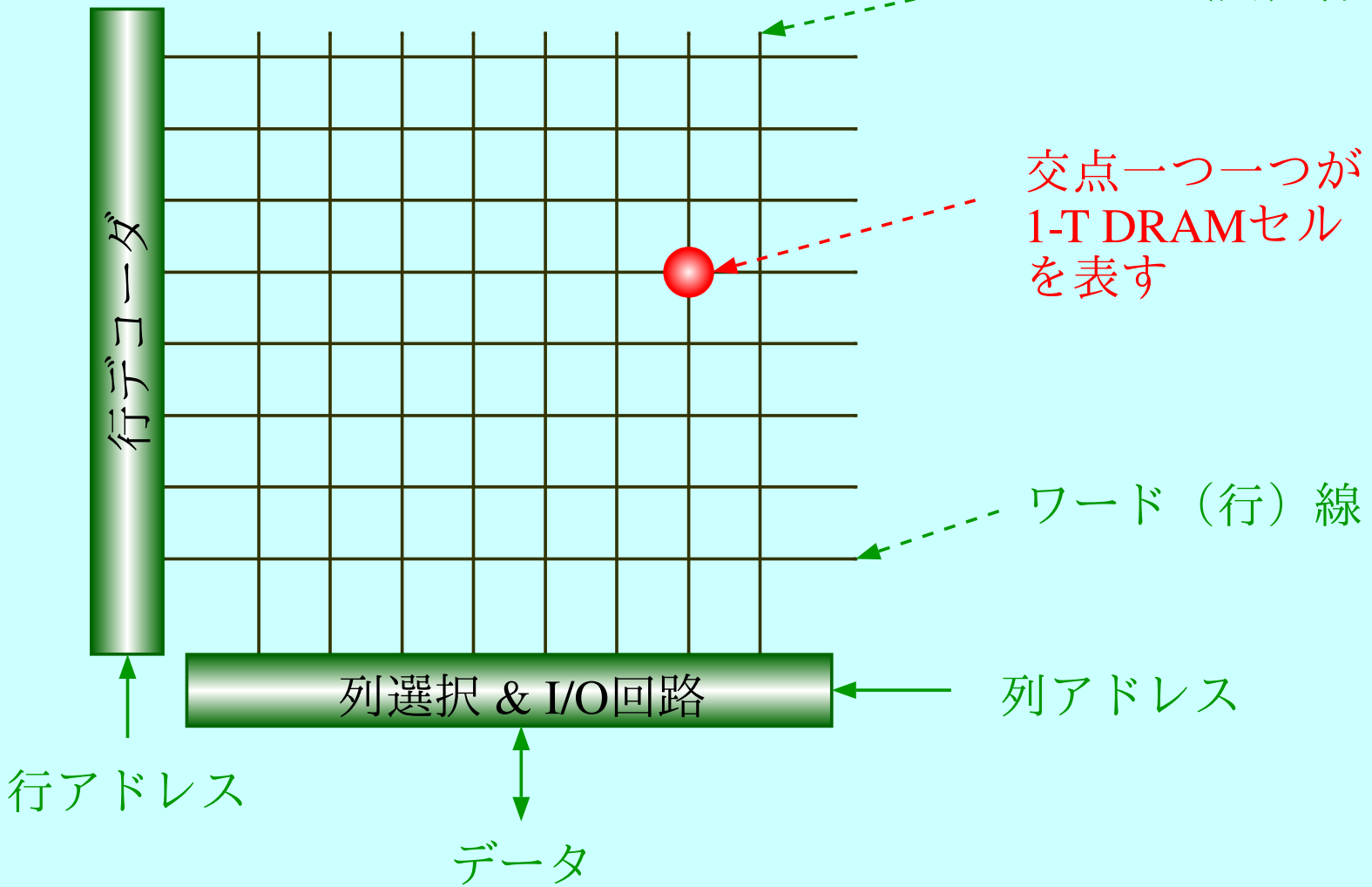
1-bit DRAM セル

1 トランジスタ DRAM セル

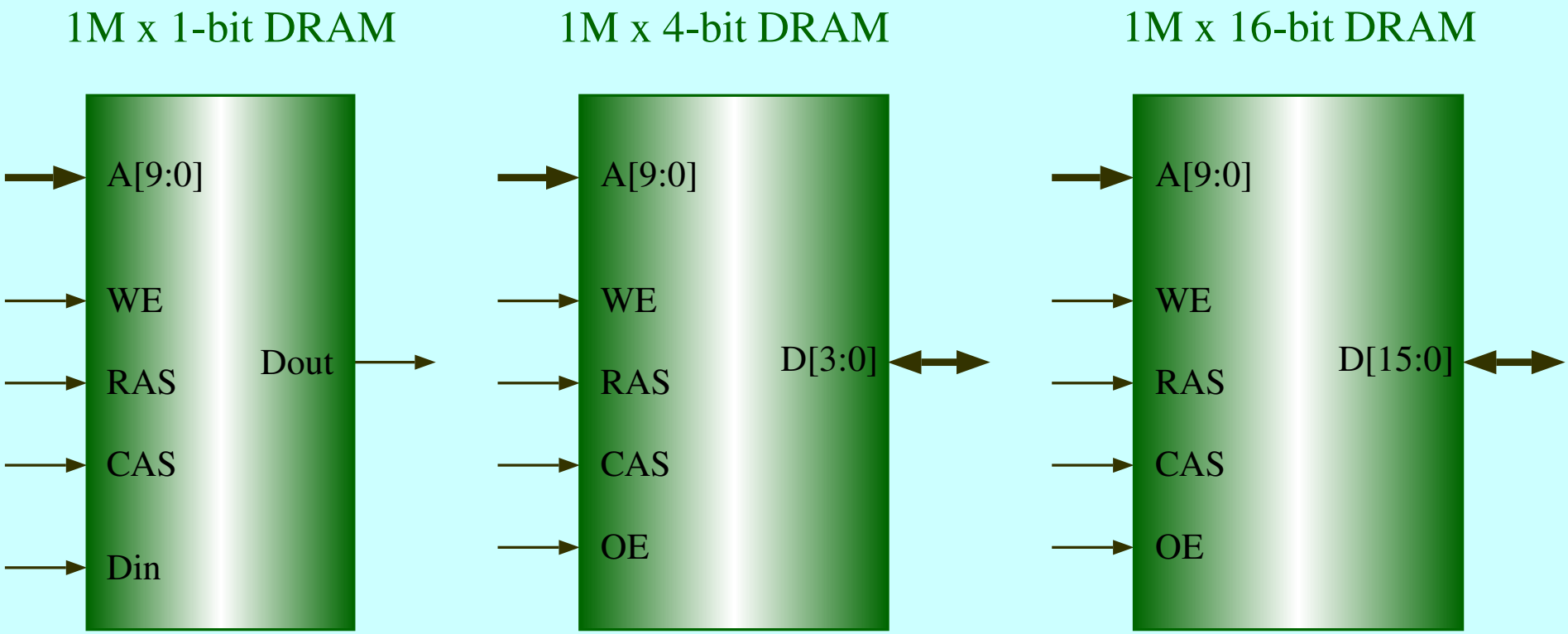


DRAMアレイ

行アドレスと列アドレスを用いて1-bitを選択する



DRAMチップの例

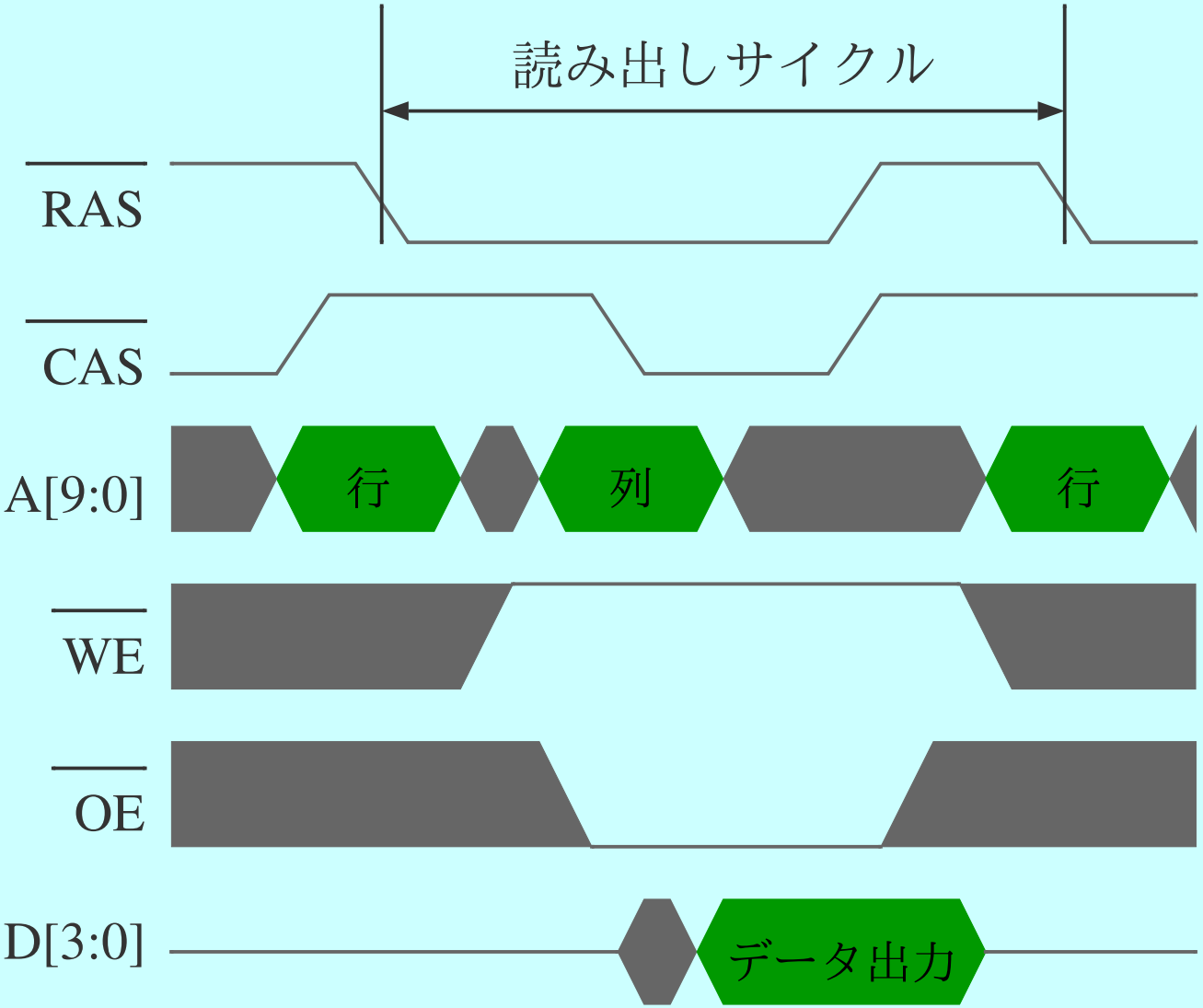


A: アドレス，D: データ，OE: 出力イネーブル（Output Enable）

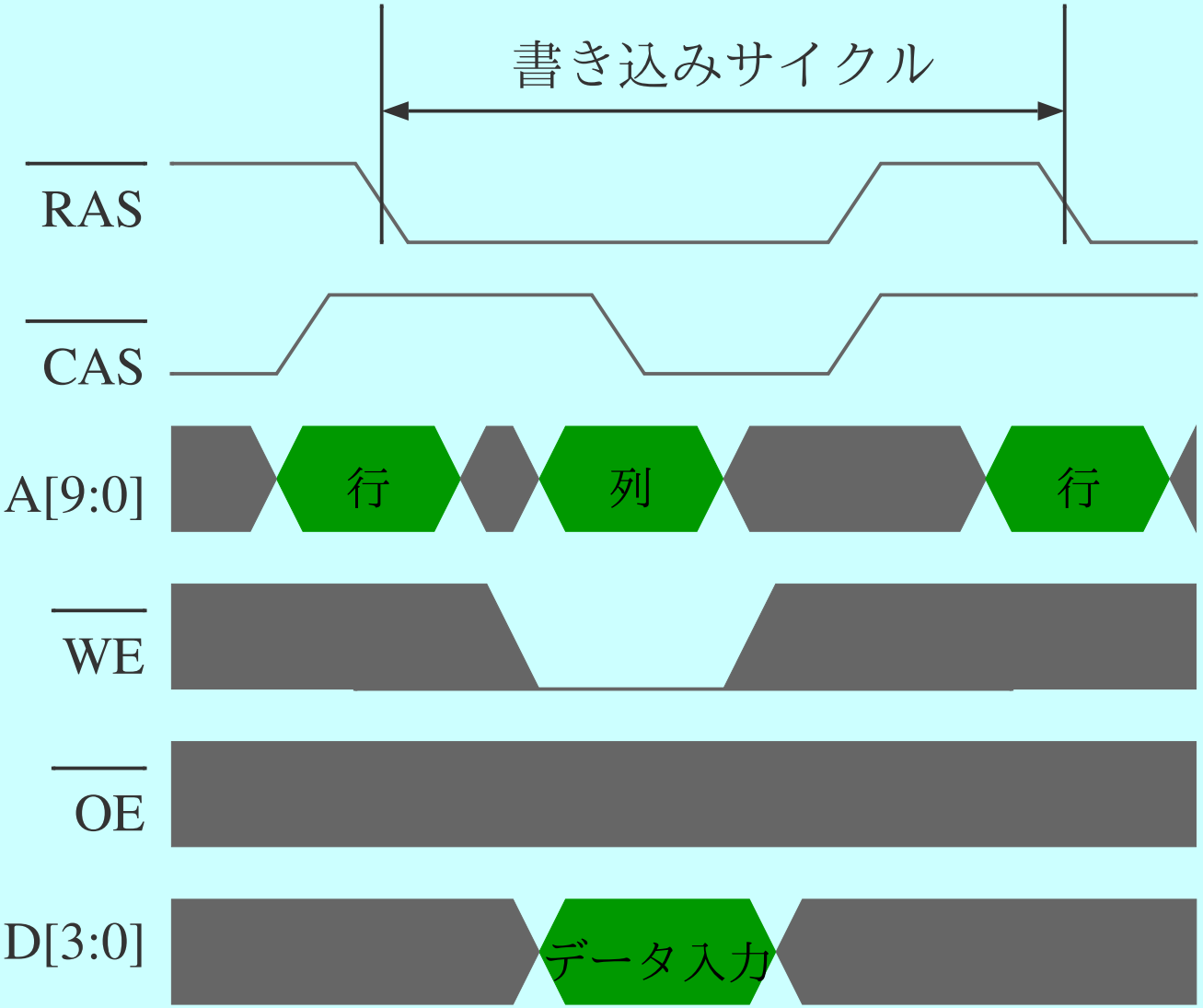
RAS: 行アドレス選択（Row Address Strobe）

CAS: 列アドレス選択（Column Address Strobe）

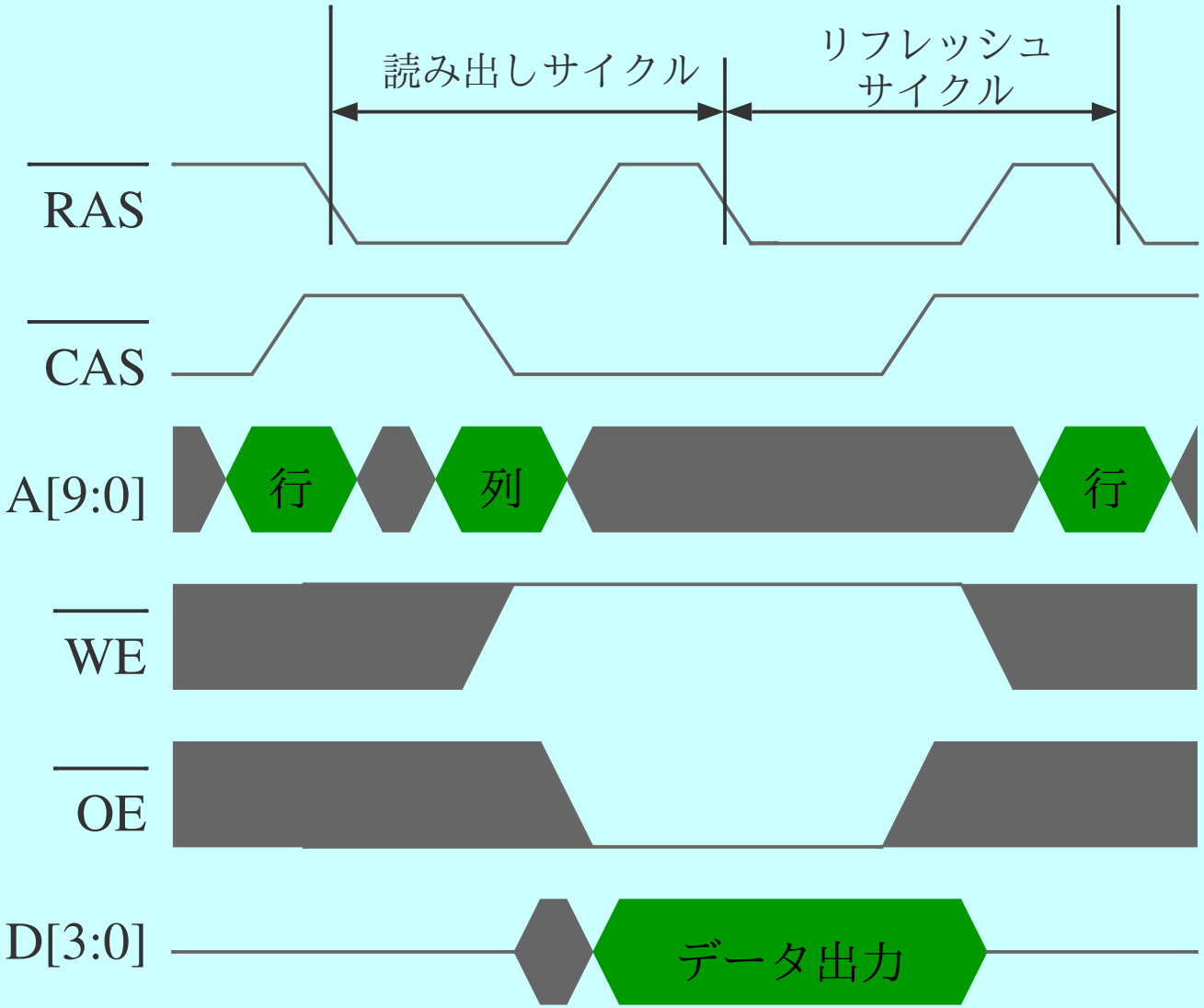
DRAM — 読み出し



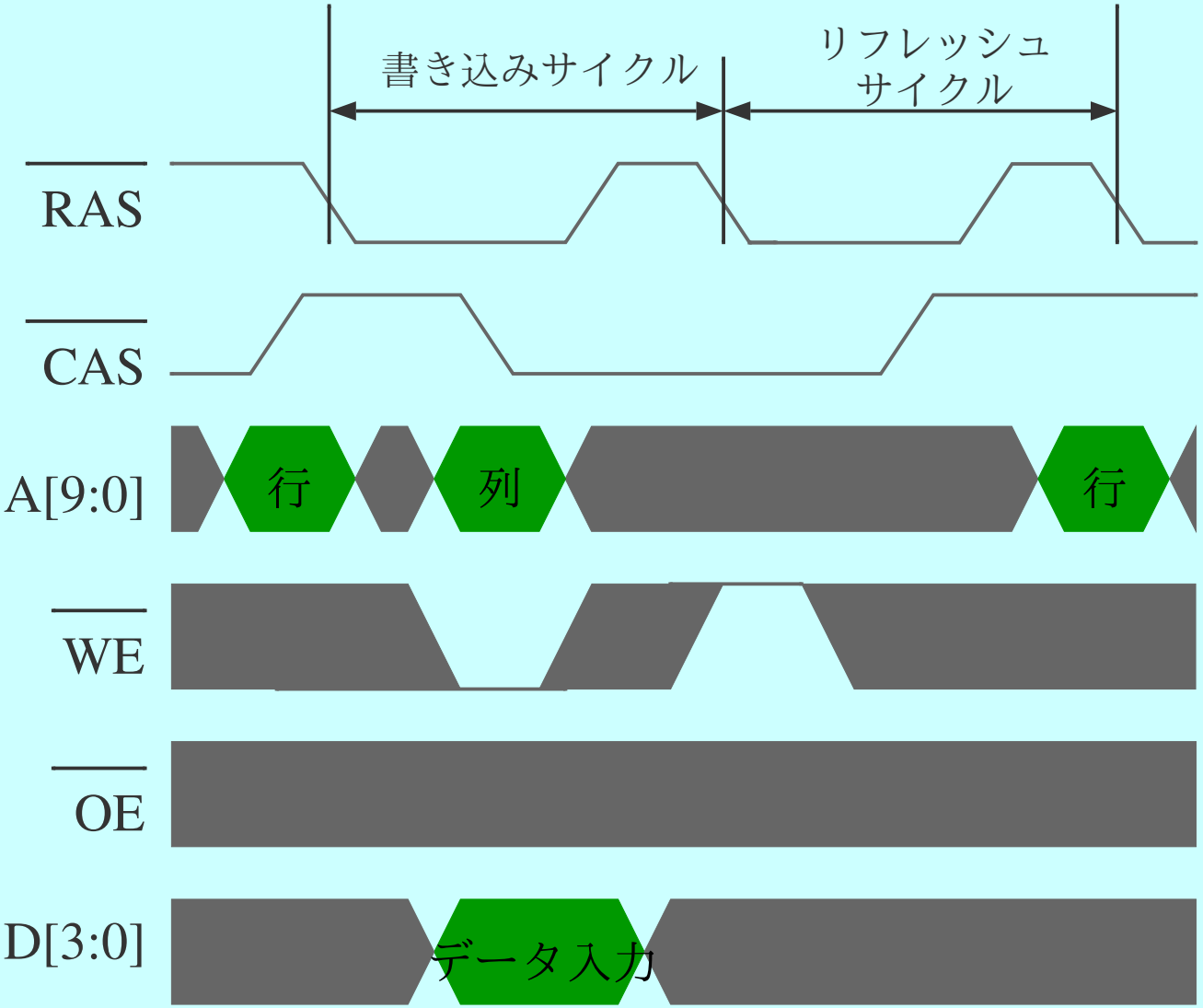
DRAM — 書き込み



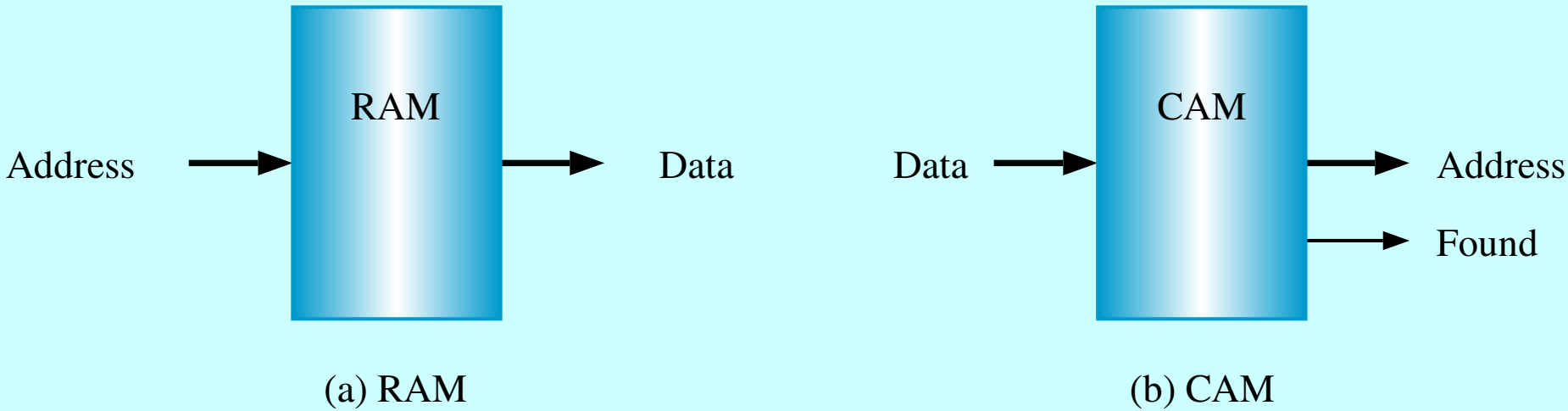
リフレッシュ・サイクル — 読み出し



リフレッシュ・サイクル — 書き込み

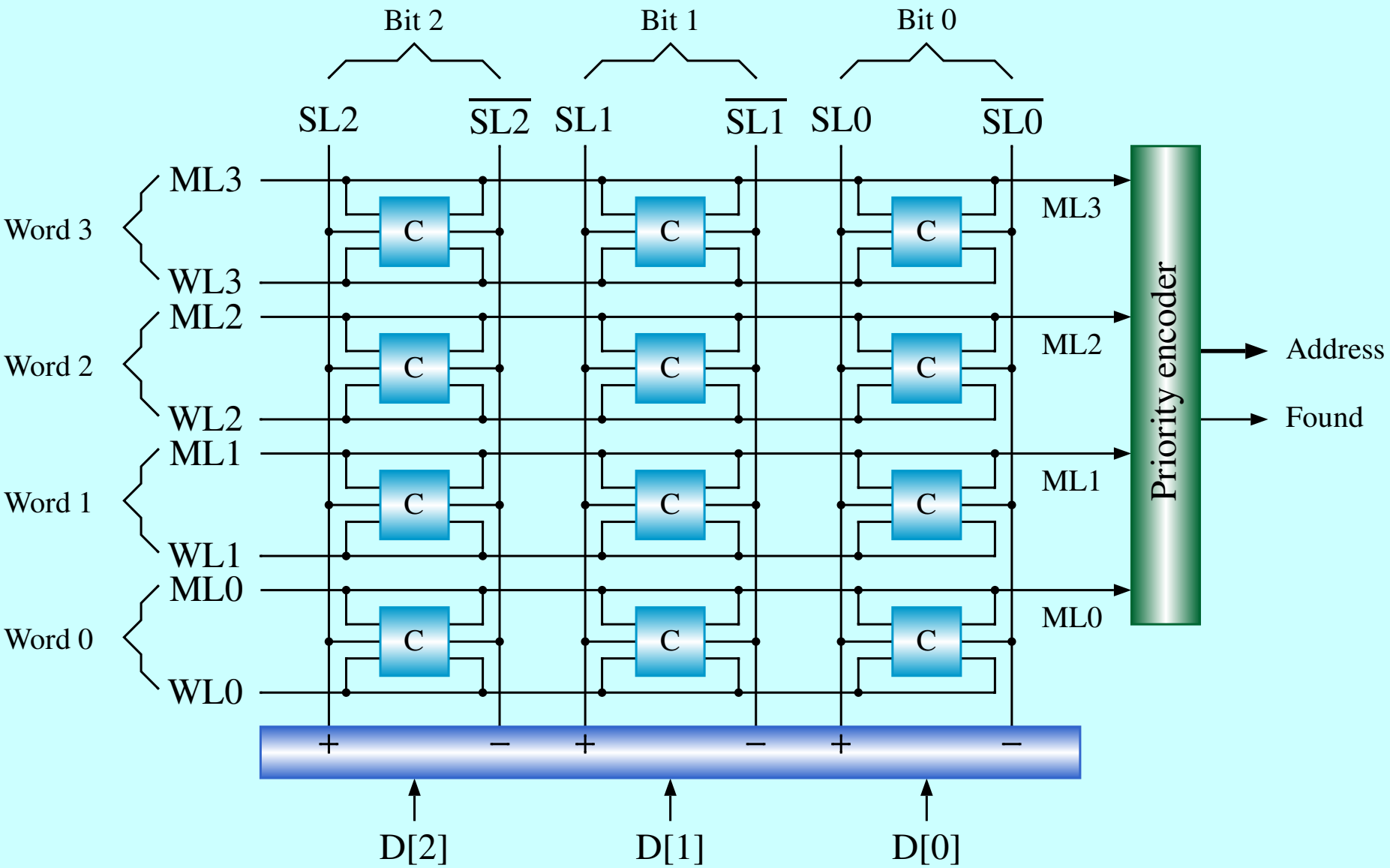


CAM (Content Addressable Memory)

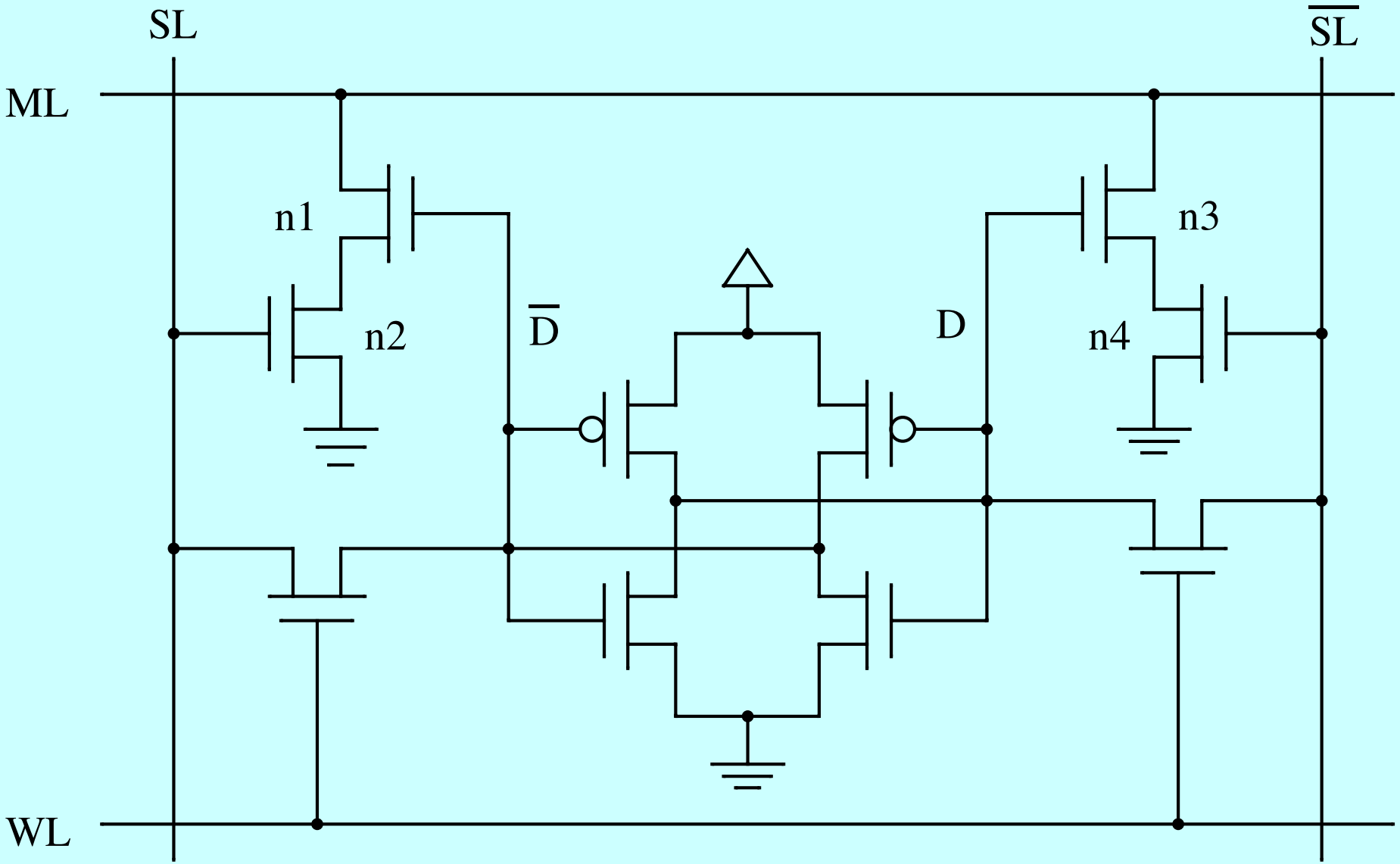


- Figure (a): In a RAM, a data word stored in a location is returned by supplying that location's address.
- Figure (b): A CAM searches the entire memory to see whether a given data word is stored anywhere in it.

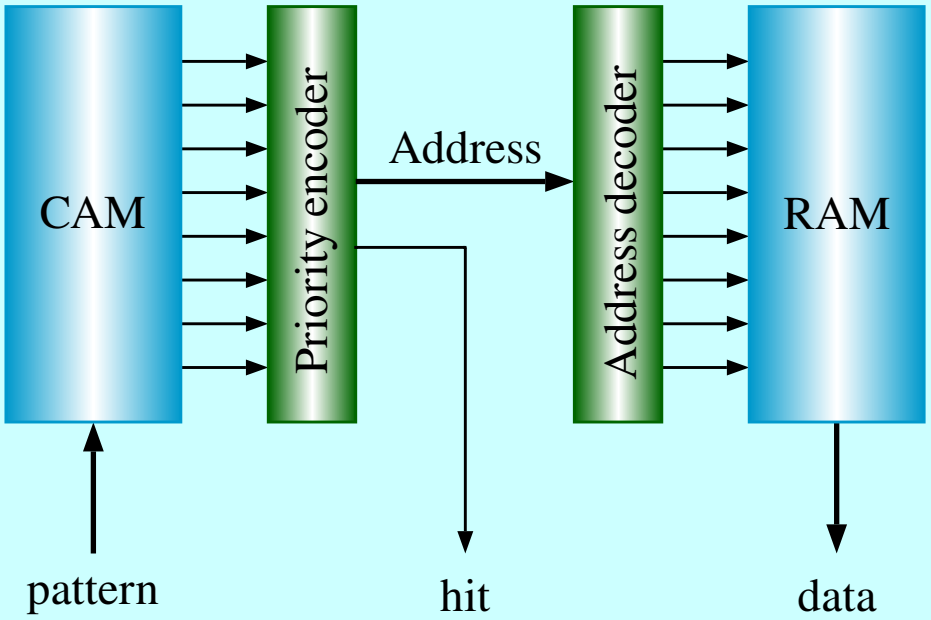
CAM Structure



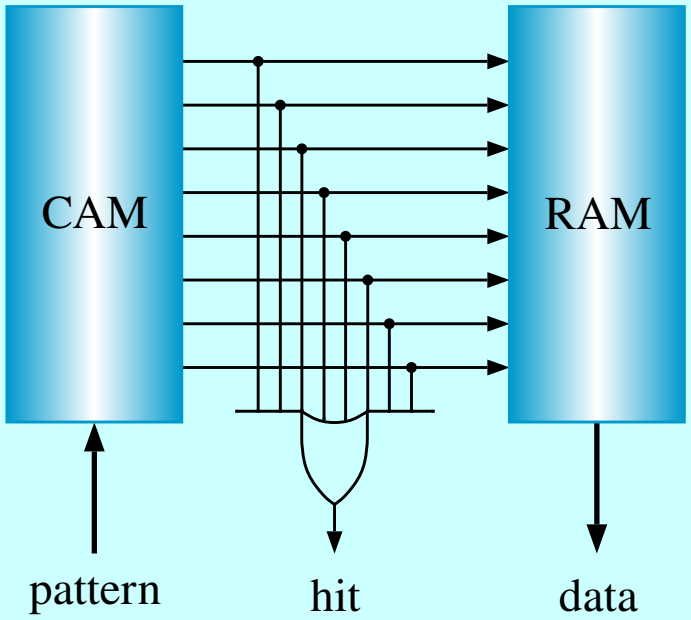
CAM Cell



Cache or TLB Built with CAM



(a) Select a data word via encoder and decoder

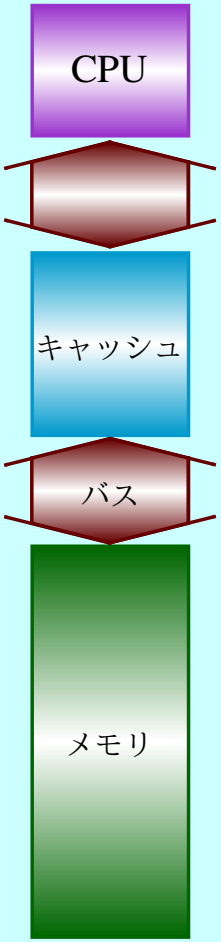


(b) Select a data word directly

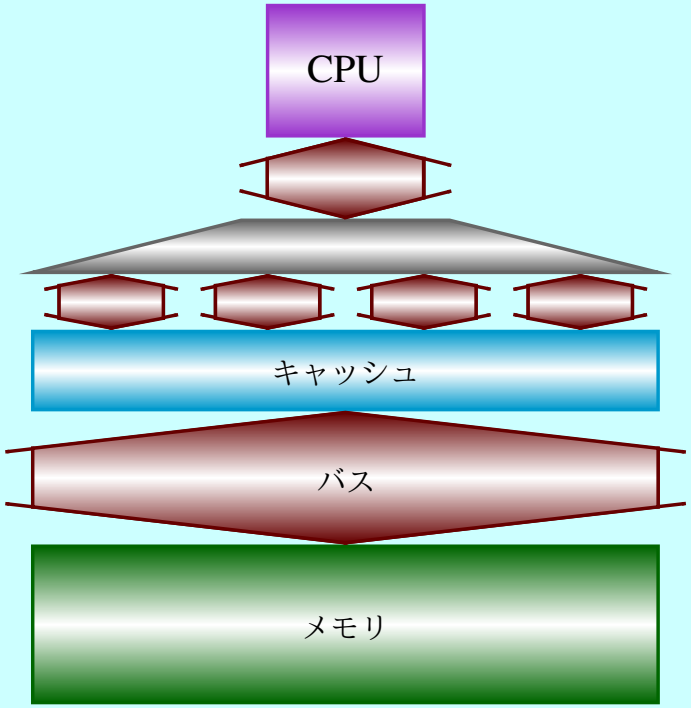
- Figure (a): Multiple matches in CAM are allowed.
- Figure (b): Multiple matches in CAM are not allowed.

メモリ幅の三つの例

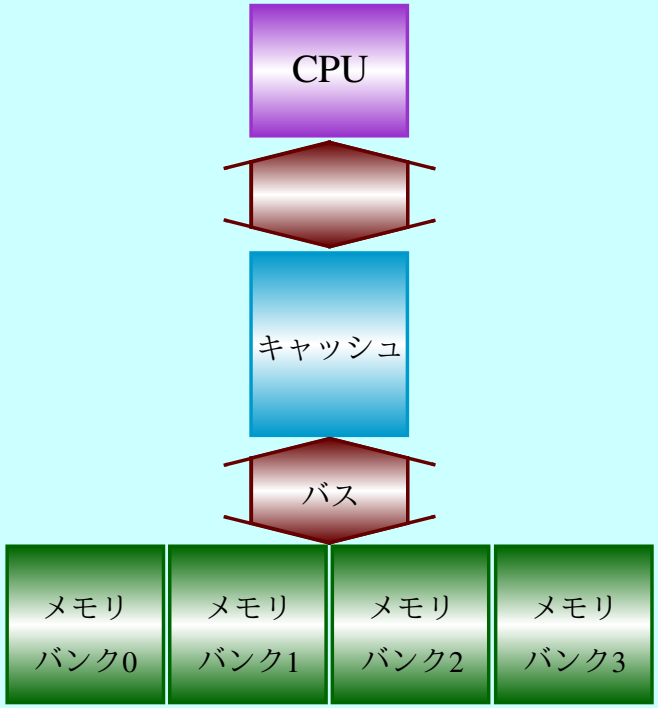
(a) 1ワード幅のメモリ構成



(b) データ幅を広げたメモリ構成

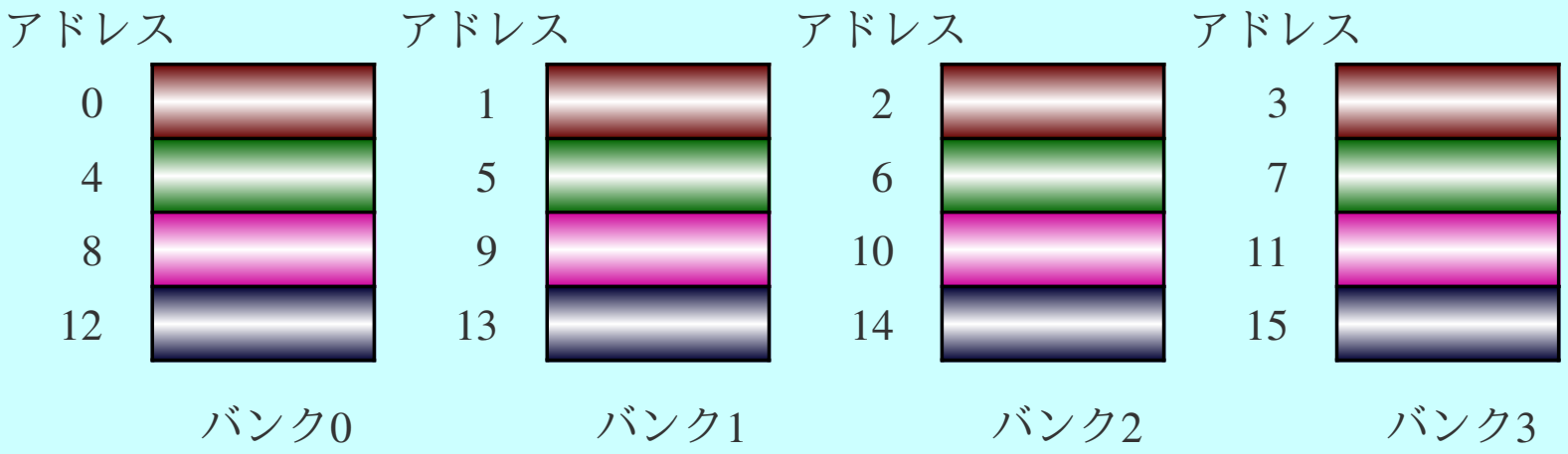


(c) インタリーブ方式のメモリ構成

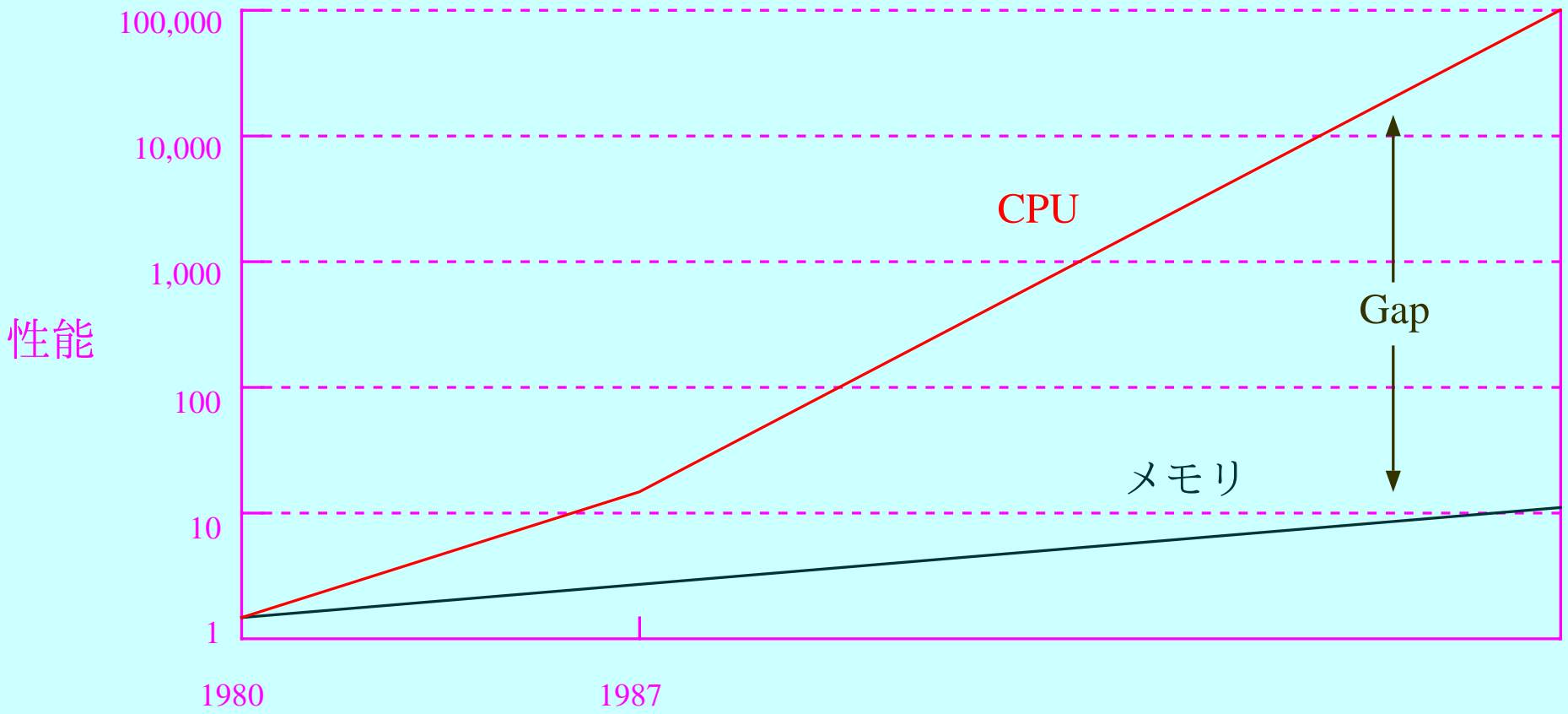


単純なインタリーブ・メモリ

- ワード単位でインタリーブされるようにメモリ・チップを複数のバンク（例えば，4 バンク）で構成することができる
- シーケンシャル・アクセス時にそのようなメモリ構成は利点を発揮する: 4 バンクがそれぞれ同時にアクセスされ，ワード・アドレスの下位 2-bit が 4 バンクから一つを選択するのに使われる



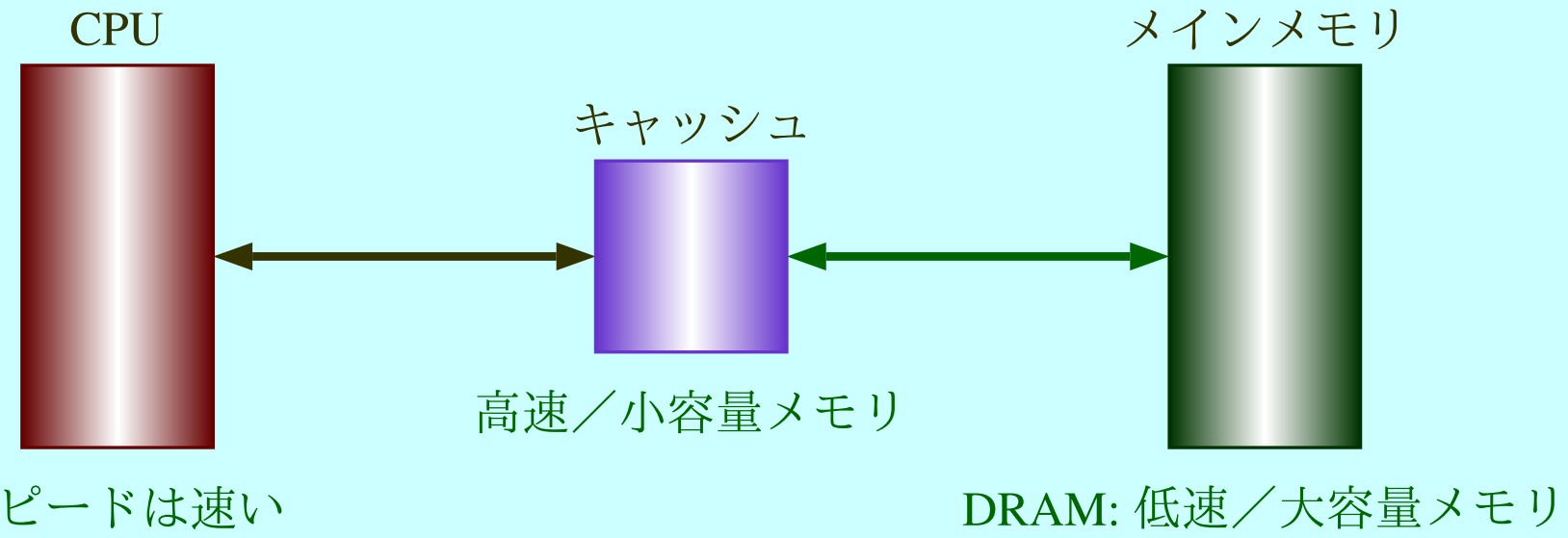
メモリとCPUの性能



CPUとメインメモリの間にキャッシュメモリを配置する

キャッシュメモリ

キャッシュは高速／小容量のメモリであり、CPU とメインメモリの間に配置されるそして、CPU がアクセスする頻度の高いコード／データをなるべくキャッシュメモリに格納しておく



CPU がメインメモリのあるアドレスからデータを読み込むとき、キャッシュにそのデータを蓄えておく。その後 CPU が再び同じアドレスからデータを読み込もうとしたら、メインメモリの代わりにキャッシュからデータを供給する

キャッシュ性能

T_c : キャッシュ・アクセス時間
T_m : メイン・メモリ・アクセス時間
h : ヒット率; $1-h$: ミス率
T : 平均メモリ・アクセス時間

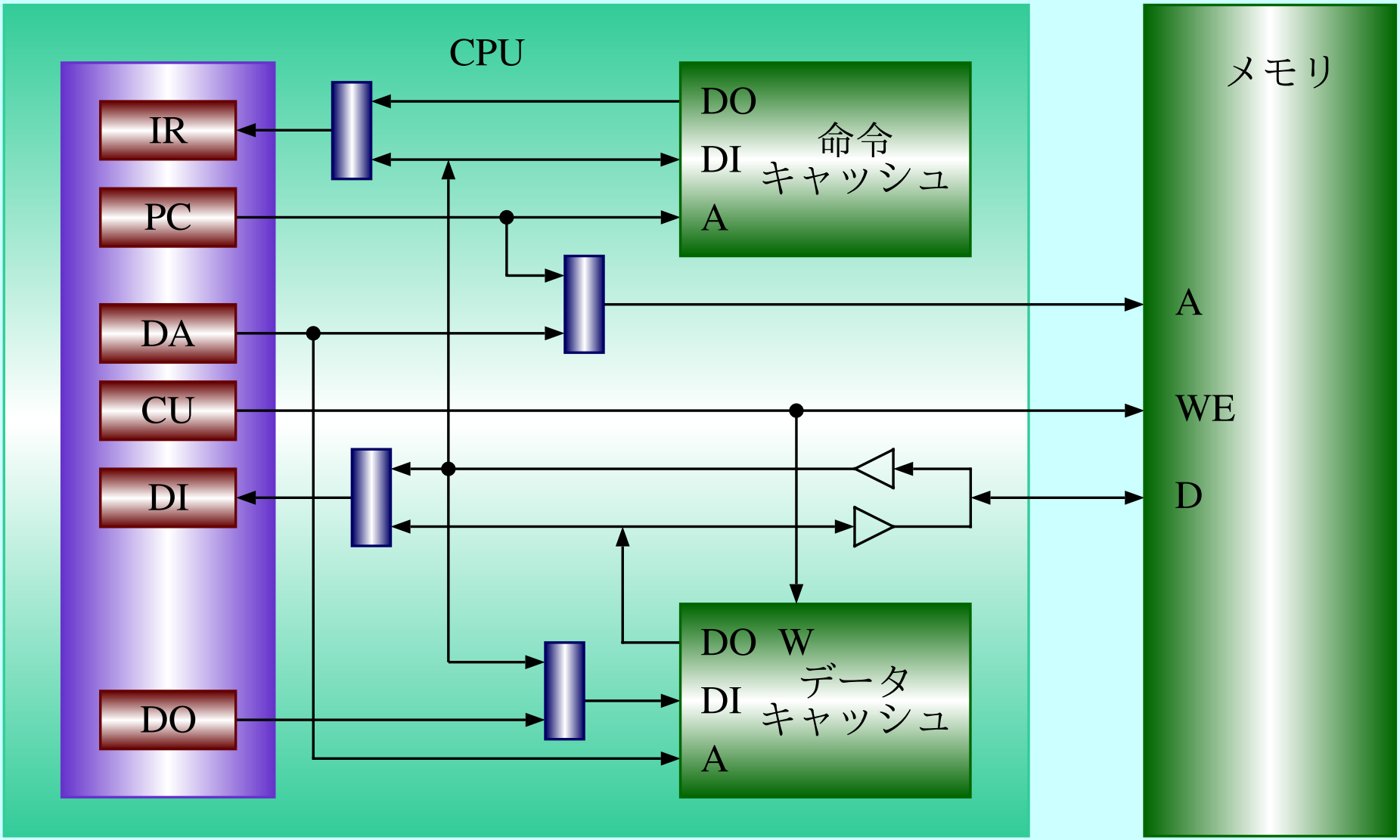
$$T = h\,T_c + (1-h)(T_c + T_m) = T_c + (1-h)\,T_m$$

例:
$T_c = 1\text{ns}$
$T_m = 50\text{ns}$
$h = 95\%$

$$T = T_c + (1-h)\,T_m = 1 + 0.05 \times 50 = 1 + 2.5 = 3.5\text{ns}$$

$$\text{Speedup} = 50/3.5 = 14.29 = 1429\%$$

オンチップ・キャッシュ



キャッシュ設計のためのパラメータ

- パラメータ
  - サイズ
    - キャッシュ・サイズ
    - ブロック・サイズ
  - アドレス
    - 物理アドレス
    - 仮想アドレス
  - 配置方式
    - ダイレクト・マップ方式
    - セット・アソシアティブ方式
    - フル・アソシアティブ方式
  - 書き込みの仕組み
    - ライト・バック
    - ライト・スルー
  - 書き込みミス時の取り扱い
    - ライト・アロケート
    - 非ライト・アロケート
  - 置き換えアルゴリズム
    - ランダム法
    - LRU 法

キャッシュ・ブロック配置

1. Direct Mapping
   ブロックを配置する場所がキャッシュ内の特定の一箇所に決まる方式をダイレクト・マップ方式と言う

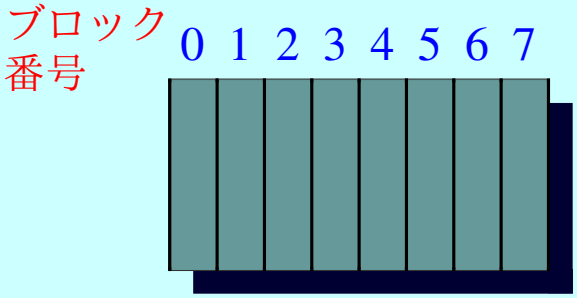
2. Fully Associative Mapping
   ブロックをキャッシュ内の任意の場所に配置できる方式をフル・アソシアティブ方式と言う

3. Set Associative Mapping
   ブロックをキャッシュ内の特定のセットに配置する方式をセット・アソシアティブ方式と言う。セットとはブロックのグループのことである。ブロックは対応するセット内の任意の場所に配置される。セット内に n 個のブロックがある場合，n ウェイ・セット・アソシアティブ方式と言う

キャッシュ・ブロック配置

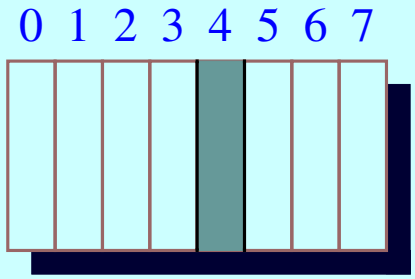
フル・アソシアティブ

12番のブロックはすべての場所に配置できる



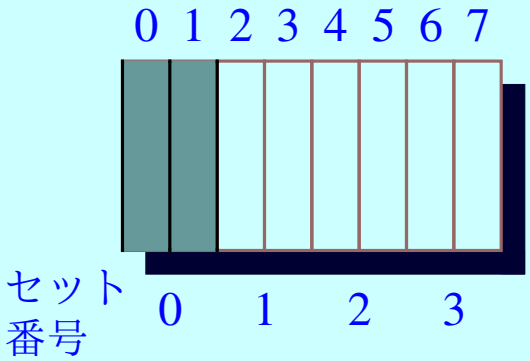
ダイレクト・マップ

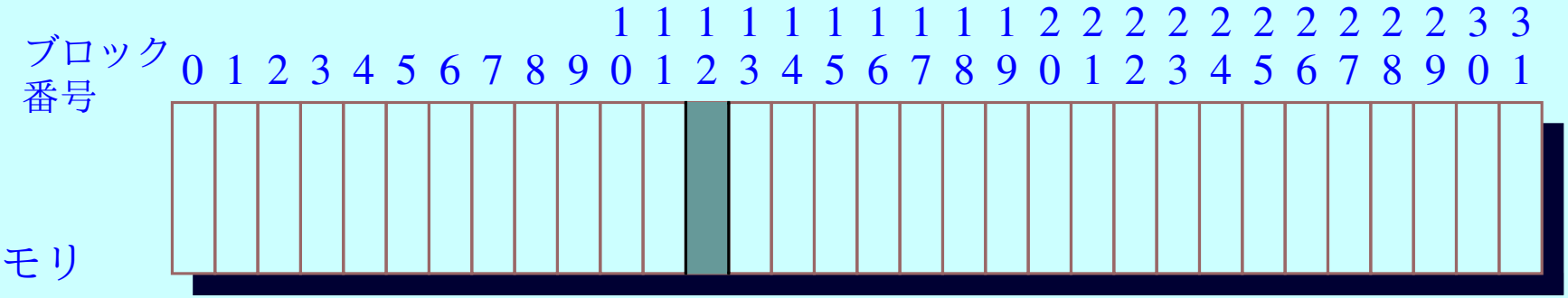
12番のブロックを配置できるのは4番目のみ（12 mod 8）



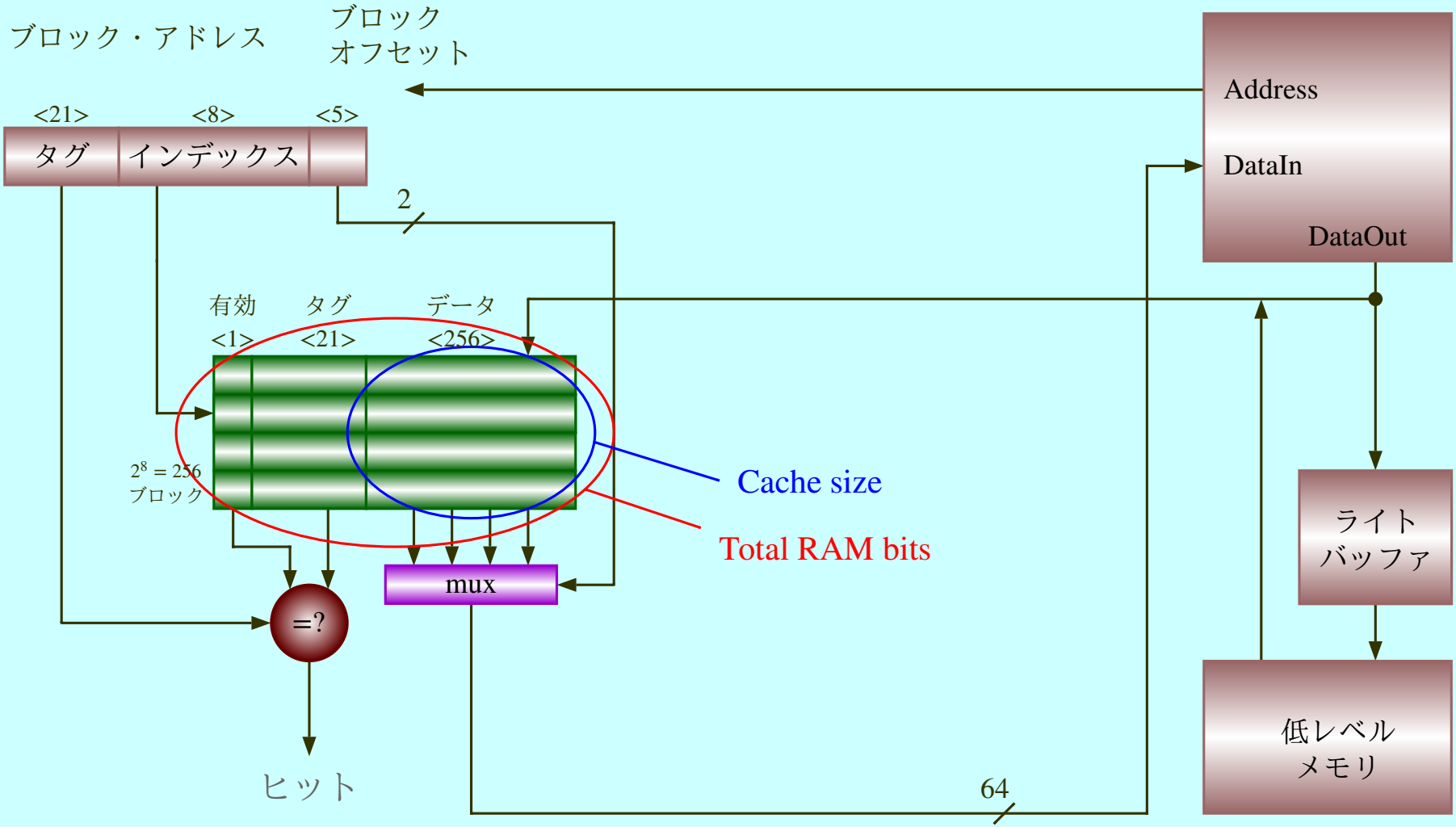
セット・アソシアティブ

12番のブロックはセット0内のすべての場所に配置できる（12 mod 4）

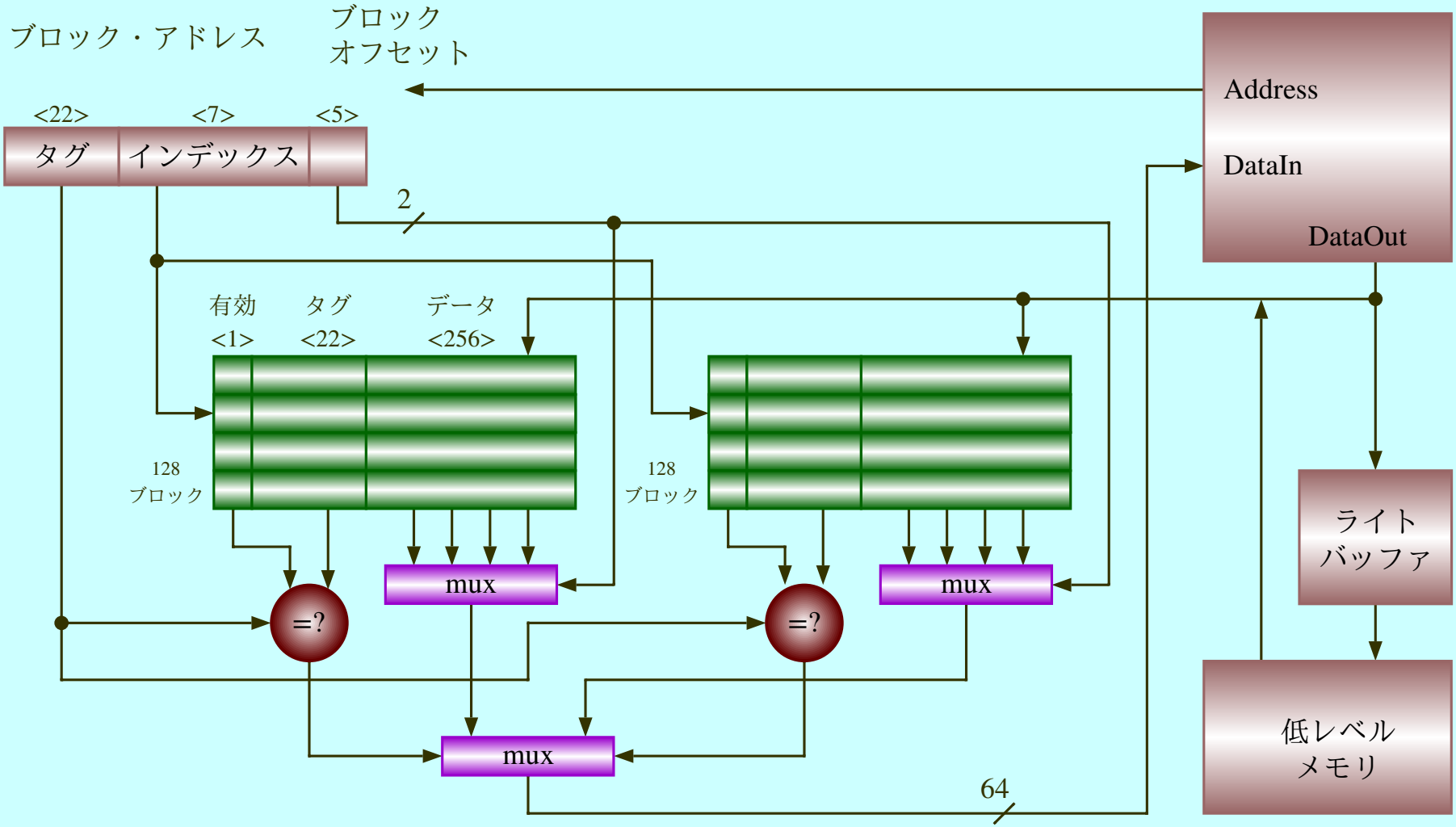




Alpha AXP 21064 キャッシュ機構



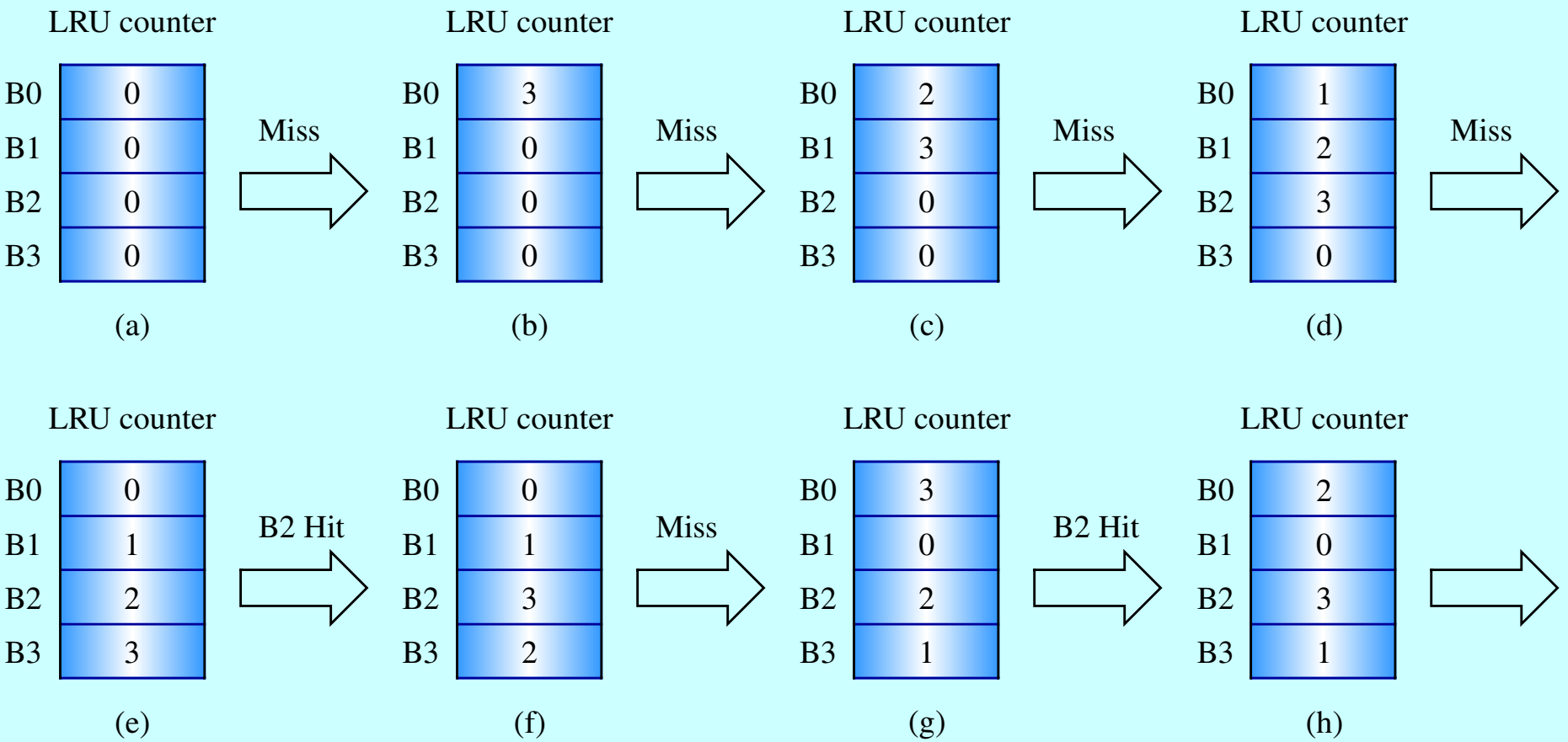
2 ウェイ・セット・アソシアティブ



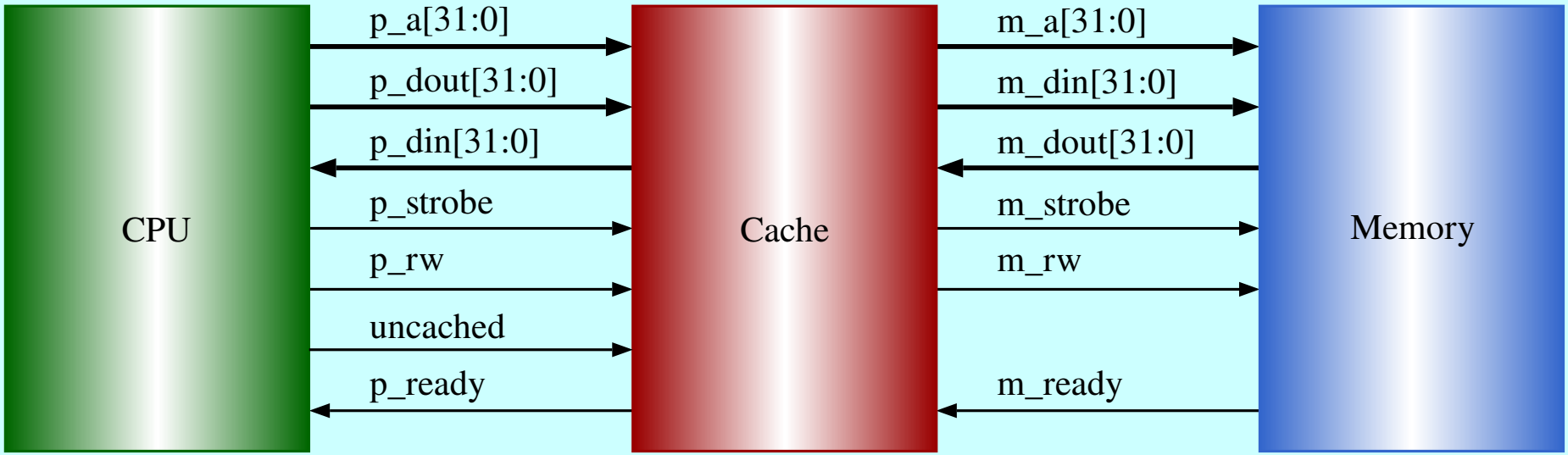
LRU Replacement Algorithm

1. Clear the counters and valid bits of all the blocks when the computer is started up.
2. If a cache miss occurs, replace a block whose counter value is a 0. If there is more than one such block, replace any of them. Set its counter to the largest value 15 (suppose there are 16 blocks in a set), and decrease all other counters whose values are not 0. We call such a counter a saturated counter.
3. If there is a hit on a block whose counter value is k, set the counter to 15, and decrease all other counters whose values are larger than k.

LRU Counter Changes

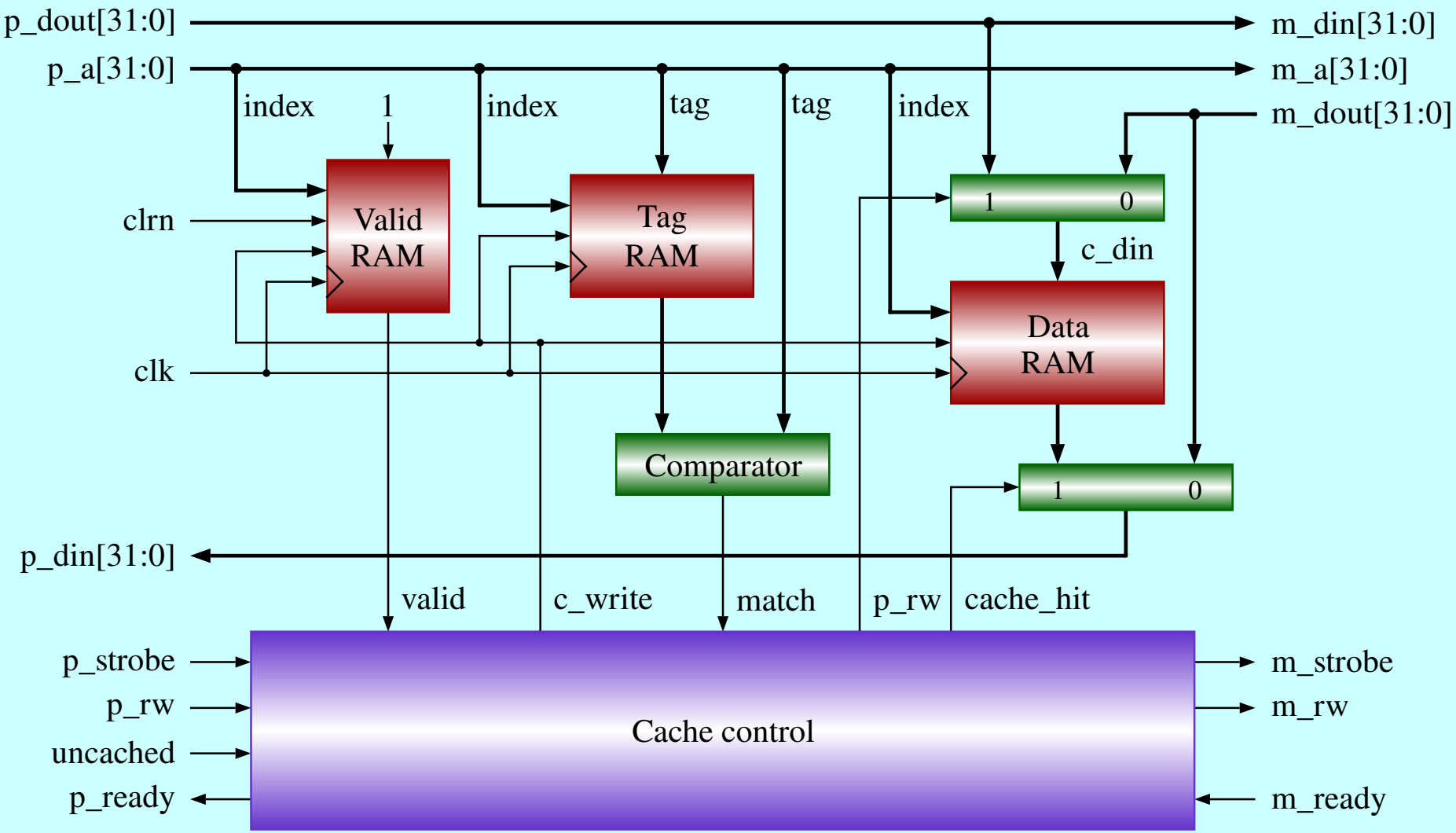


Cache in between CPU and Memory



a[]: address
dout[]: data out
din[]: data in
strobe: memory access
rw: read or write
m_ready: memory ready
p_ready: ready for processor

Block Diagram of Write Through Cache



d_cache.v

```
‘timescale 1ns/1ns
module d_cache (  // direct mapping, 2^6 blocks, 1 word/block, write-through
    input  [31:0] p_a,                          // cpu address
    input  [31:0] p_dout,                       // cpu data out  to mem
    output [31:0] p_din,                        // cpu data in from mem
    input         p_strobe,                     // cpu strobe
    input         p_rw,                         // cpu read/write command
    input         uncached,                     // uncached
    output        p_ready,                      // ready (to cpu)
    input         clk, clrn,                    // clock and reset
    output [31:0] m_a,                          // mem address
    input  [31:0] m_dout,                       // mem data out  to cpu
    output [31:0] m_din,                        // mem data in from cpu
    output        m_strobe,                     // mem strobe
    output        m_rw,                         // mem read/write
    input         m_ready);                     // mem ready

    reg           d_valid [0:63];               // 1-bit valid
    reg    [23:0] d_tags  [0:63];               // 24-bit tag
    reg    [31:0] d_data  [0:63];               // 32-bit data
    wire   [23:0] tag = p_a[31:8];              // address tag
    wire   [31:0] c_din;                        // data to cache
    wire    [5:0] index = p_a[7:2];             // block index
    wire          c_write;                      // cache write

    integer       i;
```

d_cache.v

```
    always @ (posedge clk or negedge clrn)
        if (!clrn) begin
            for (i=0; i<64; i=i+1)
                d_valid[i] <= 0;                    // clear valid
        end else if (c_write)
            d_valid[index] <= 1;                    // write valid
    always @ (posedge clk)
        if (c_write) begin
            d_tags[index] <= tag;                   // write address tag
            d_data[index] <= c_din;                 // write data
        end
    wire          valid = d_valid[index];           // read cache valid
    wire   [23:0] tagout = d_tags[index];           // read cache tag
    wire   [31:0] c_dout = d_data[index];           // read cache data
    wire cache_hit  = p_strobe &   valid & (tagout == tag);    // cache hit
    wire cache_miss = p_strobe & (!valid | (tagout != tag));   // cache miss
    assign m_din    = p_dout;                       // mem <-- cpu data
    assign m_a      = p_a;                          // mem <-- cpu address
    assign m_rw     = p_rw;                         // write through
    assign m_strobe = p_rw | cache_miss;            // also read on miss
    assign p_ready  = ~p_rw & cache_hit |           // read and hit or
                      (cache_miss | p_rw) & m_ready;  // write and mem ready
    assign c_write  = ~uncached & (p_rw | cache_miss & m_ready);    // write
    assign c_din    = p_rw?      p_dout : m_dout;   // data from cpu or mem
    assign p_din    = cache_hit? c_dout : m_dout;   // data from cache or mem
endmodule                                                          d_cache.v
```

d_cache_tb.v

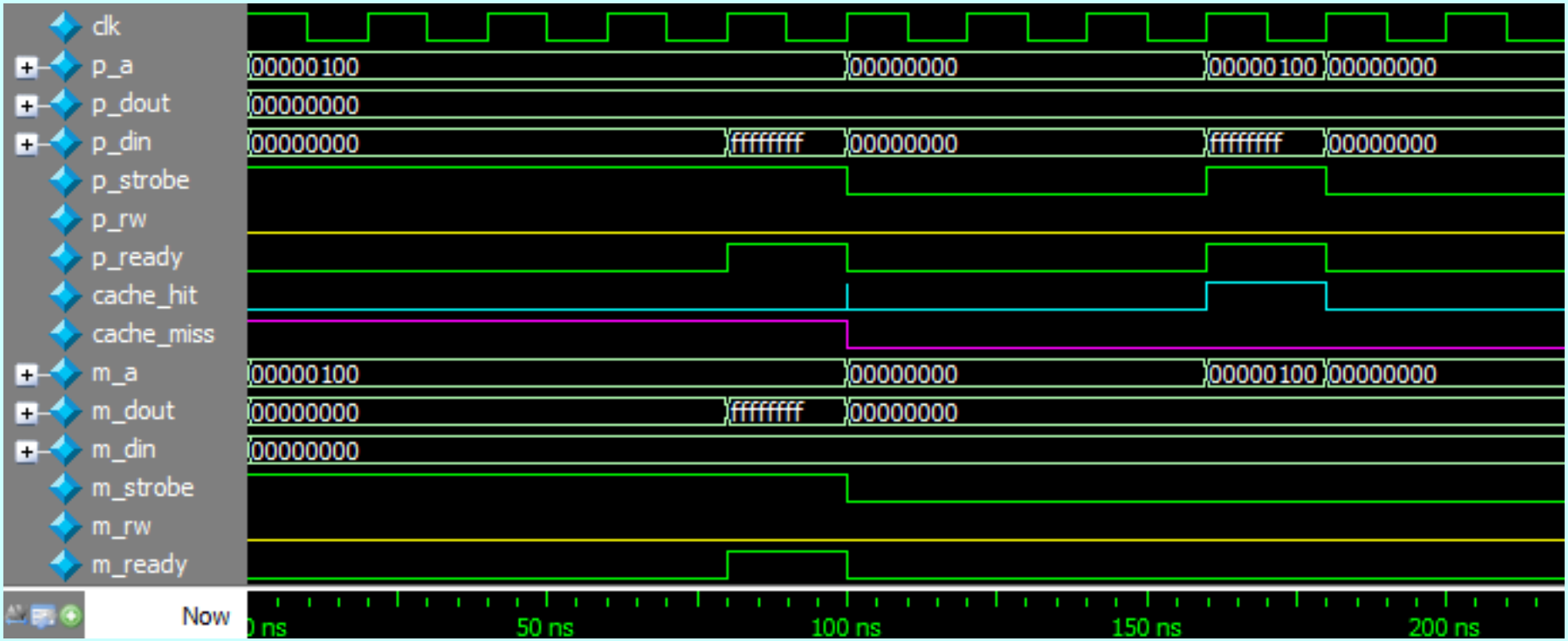
```
‘timescale 1ns/1ps
module d_cache_tb;
    reg  [31:0] p_a, p_dout, m_dout;
    reg         p_strobe, p_rw, uncached, clk,clrn, m_ready;
    wire [31:0] p_din, m_a, m_din;
    wire        p_ready, m_strobe, m_rw;

    d_cache dcache (p_a,p_dout,p_din,p_strobe,p_rw,uncached,p_ready,
                    clk,clrn,m_a,m_dout,m_din,m_strobe,m_rw,m_ready);
    initial begin
             clrn     = 0;
             clk      = 1;
             p_a      = 32’h00000100;
             p_dout   = 32’h00000000;
             p_strobe = 1;
             p_rw     = 0;
             m_dout   = 32’h00000000;
             uncached = 0;
             m_ready  = 0;
        #10.001 clrn  = 1;
        #70  m_dout   = 32’hffffffff;
             m_ready  = 1;
        #20  p_a      = 32’h00000000;
             p_strobe = 0;
             m_dout   = 32’h00000000;
             m_ready  = 0;
```

d_cache_tb.v

```
        #60  p_a      = 32'h00000100;
             p_strobe = 1;
        #20  p_a      = 32'h00000000;
             p_strobe = 0;
        #80  p_a      = 32'h00000104;
             p_dout   = 32'h5555aaaa;
             p_rw     = 1;
             p_strobe = 1;
        #100 m_ready  = 1;
        #20  p_a      = 32'h00000000;
             p_dout   = 32'h00000000;
             p_strobe = 0;
             p_rw     = 0;
             m_ready  = 0;
        #20  p_a      = 32'h00000104;
             p_strobe = 1;
        #20  p_a      = 32'h00000000;
             p_strobe = 0;
        #20  $finish;
    end
    always #10 clk = !clk;
    initial begin
        $dumpfile ("d_cache.vcd");
        $dumpvars;
    end
endmodule
```

d_cache_tb.v

Waveform of Data Cache Read



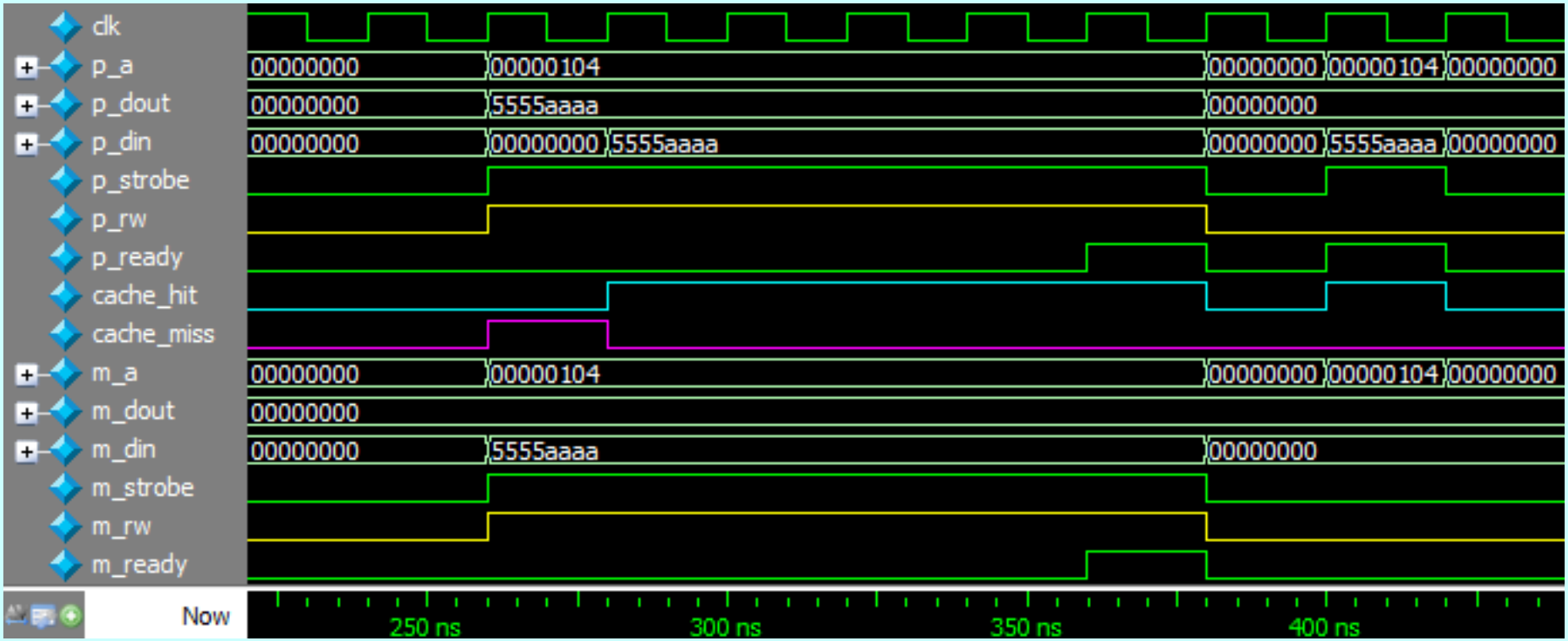
000ns: Address = 0x100, memory read, cache miss

080ns: Memory data ready, write cache

100ns - 160ns: Do not access memory (strobe = 0)

160ns: Address = 0x100, memory read, cache hit

Waveform of Data Cache Write



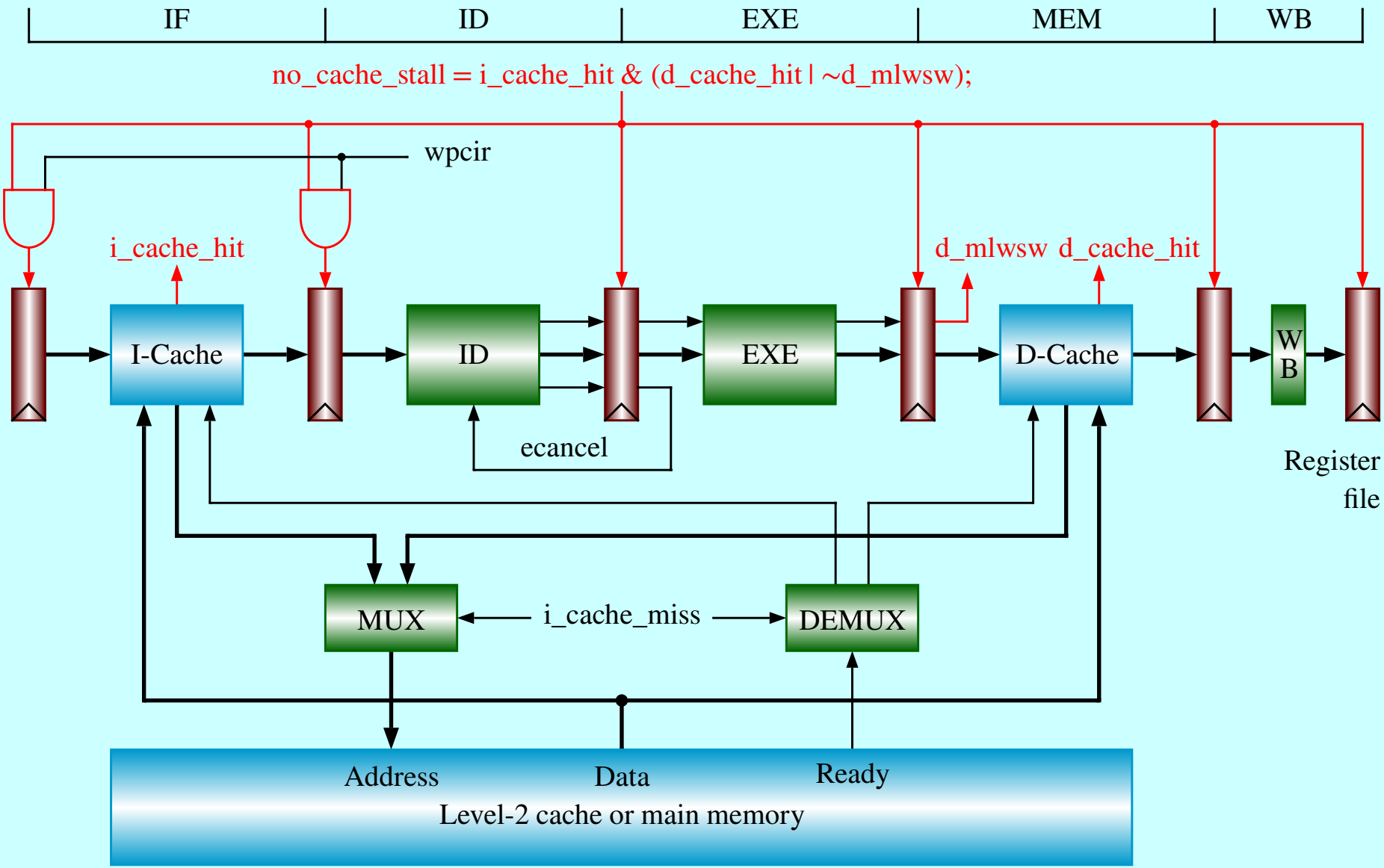
260ns: Address = 0x104, memory write, cache miss

280ns: Write cache; 360ns: write memory ready

380ns - 400ns: Do not access memory (strobe = 0)

400ns: Address = 0x104, memory read, cache hit

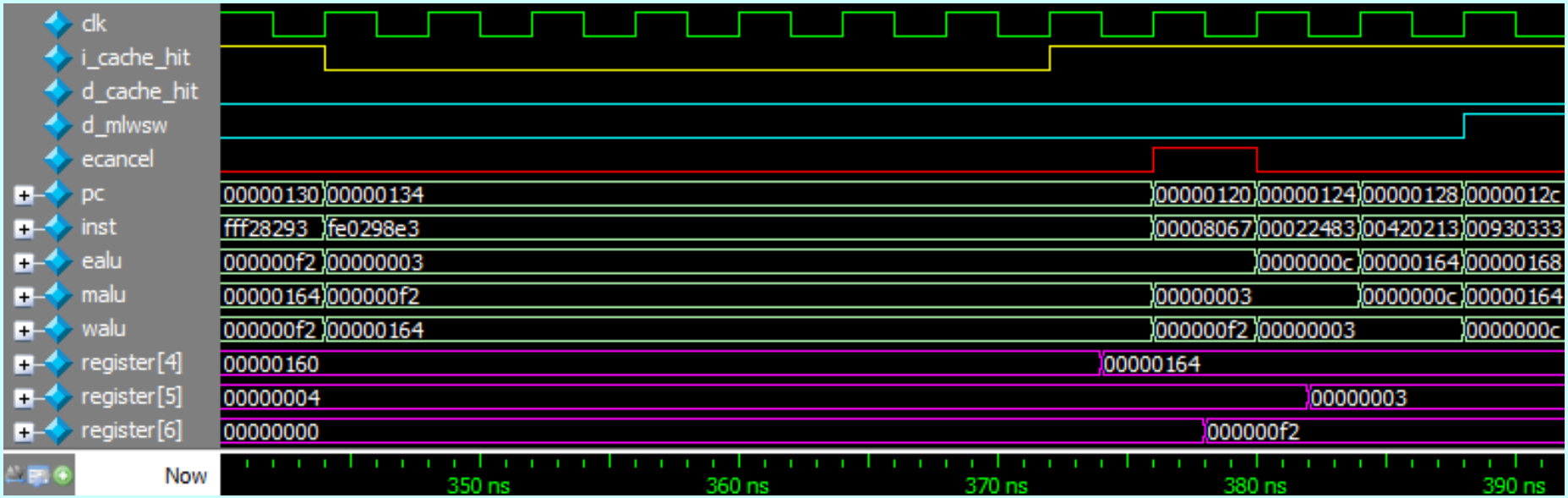
Pipelined CPU + Caches



Test Program

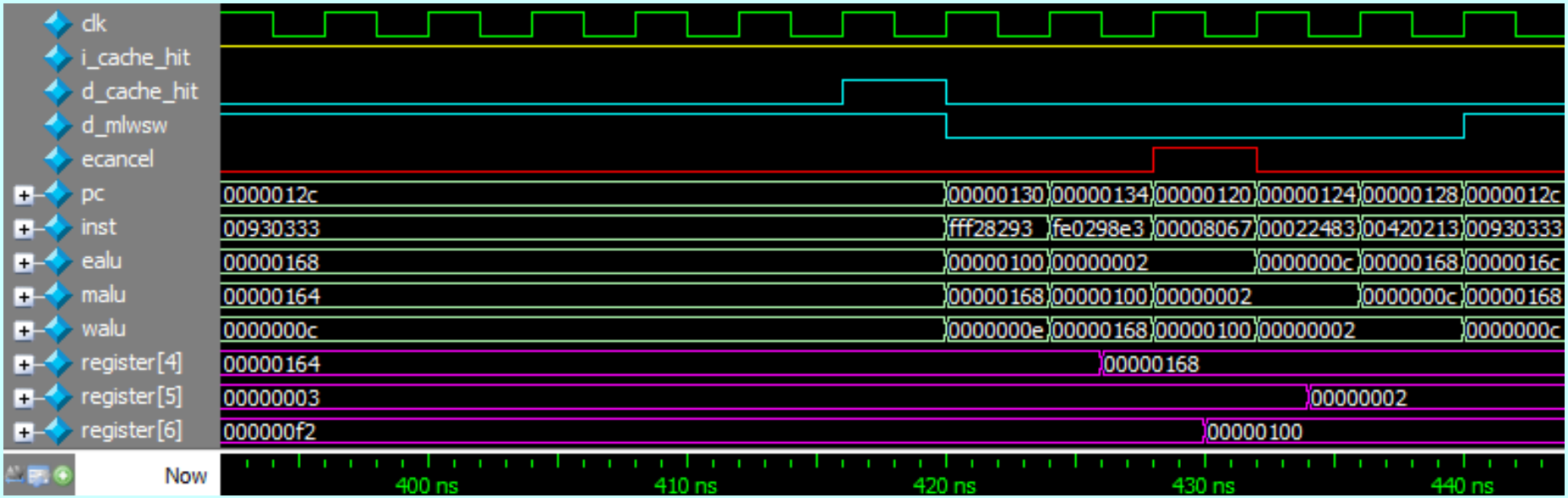
```
.text
main: # ...                        ( PC)
sum:  add     x6, x0, x0    # (11c) x6 <- 0 (subroutine entry)
loop: lw      x9, 0(x4)     # (120) x9 <- memory[x4+0]
      addi    x4, x4,  4    # (124) x4 <- x4 + 4 (address+4)
      add     x6, x6, x9    # (128) x6 <- x6 + x9 (sum)
      addi    x5, x5, -1    # (12c) x5 <- x5 - 1 (counter--)
      bne     x5, x0, loop  # (130) if x5 != 0, goto loop
      ret     x1            # (134) return from subroutine 100
.data
.word 0x000000f2  # a[0] 0xf2
.word 0x0000000e  # a[1] 0xf2  + 0xe   = 0x100
.word 0x00000200  # a[2] 0x100 + 0x200 = 0x300
.word 0xffffffff  # a[3] 0x300 + (-1)  = 0x2ff
```

Waveform of Instruction Cache Miss



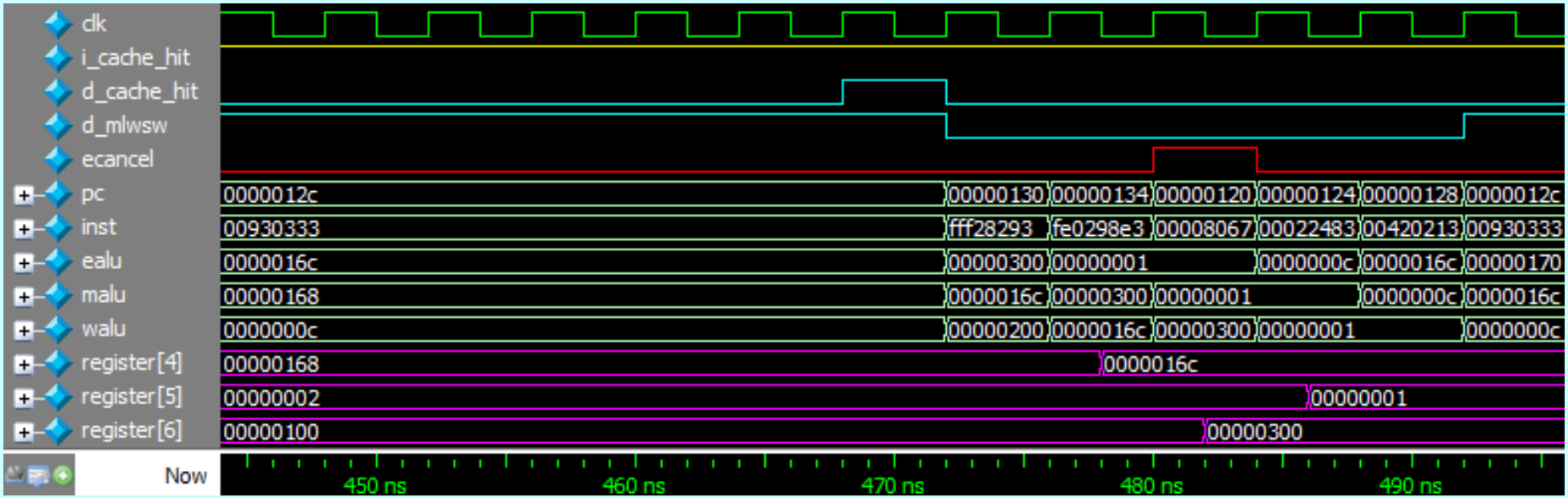

```
                                ( PC)
      bne     x5, x0, loop # (130) inst cache hit
      ret     x1           # (134) inst cache miss
loop: lw      x9, 0(x4)    # (120) inst cache hit (lw in IF)
      addi    x4, x4,  4   # (124) inst cache hit (lw in ID)
      add     x6, x6, x9   # (128) inst cache hit (lw in EXE)
      addi    x5, x5, -1   # (12c) inst cache hit (lw in MEM)
```

Waveform of Data Cache Miss



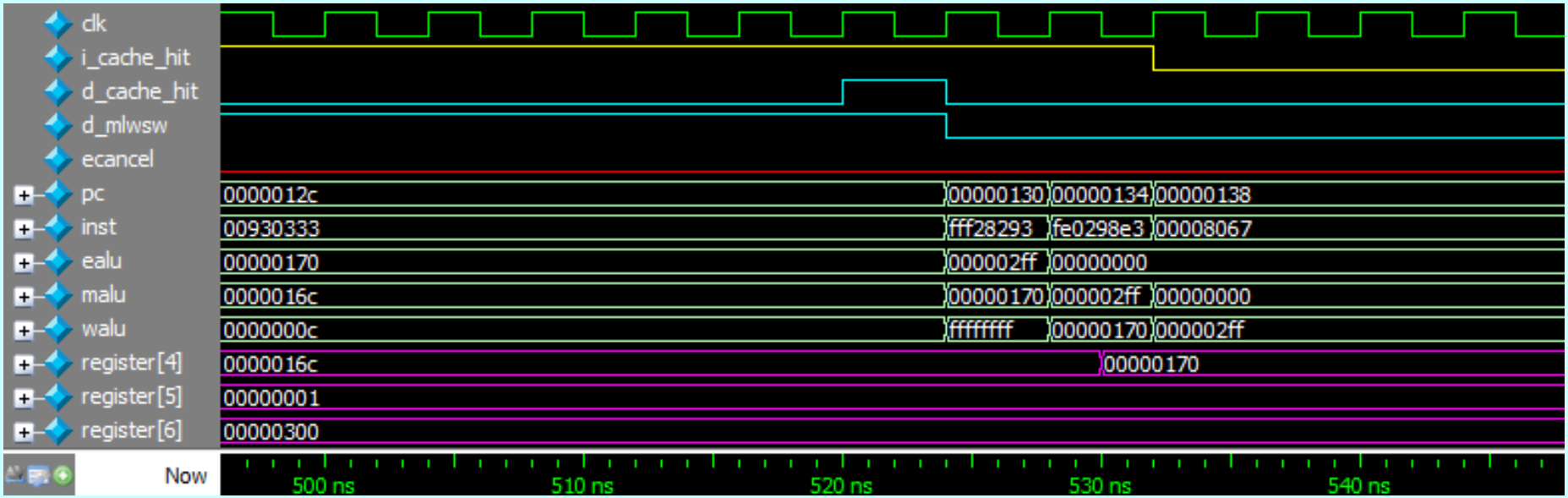

```
      addi    x5, x5, -1    # (12c) data cache miss(lw in MEM)
      bne     x5, x0, loop  # (130) inst cache hit
      ret     x1            # (134) inst cache hit, canceled
loop: lw      x9, 0(x4)     # (120) inst cache hit (lw in IF)
      addi    x4, x4,  4    # (124) inst cache hit (lw in ID)
      add     x6, x6, x9    # (128) inst cache hit (lw in EXE)
      addi    x5, x5, -1    # (12c) inst cache hit (lw in MEM)
```

Waveform of Data Cache Miss



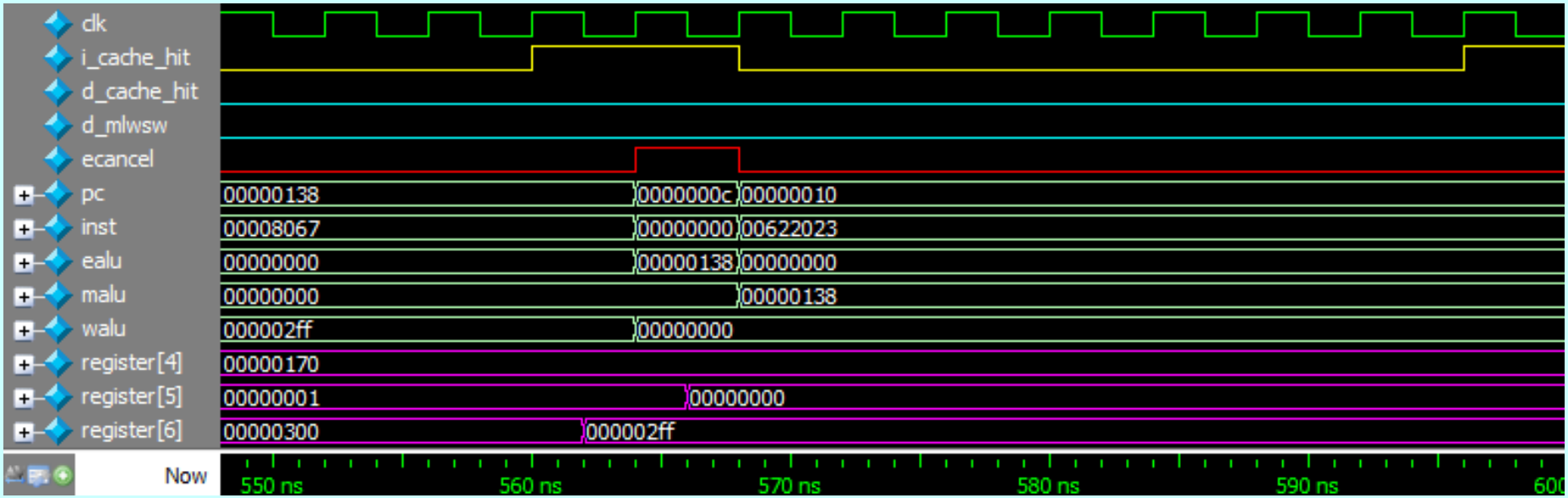

```
      addi    x5, x5, -1   # (12c) data cache miss(lw in MEM)
      bne     x5, x0, loop # (130) inst cache hit
      ret     x1           # (134) inst cache hit, canceled
loop: lw      x9, 0(x4)    # (120) inst cache hit (lw in IF)
      addi    x4, x4,  4   # (124) inst cache hit (lw in ID)
      add     x6, x6, x9   # (128) inst cache hit (lw in EXE)
      addi    x5, x5, -1   # (12c) inst cache hit (lw in MEM)
```

Waveform of Data Cache Miss



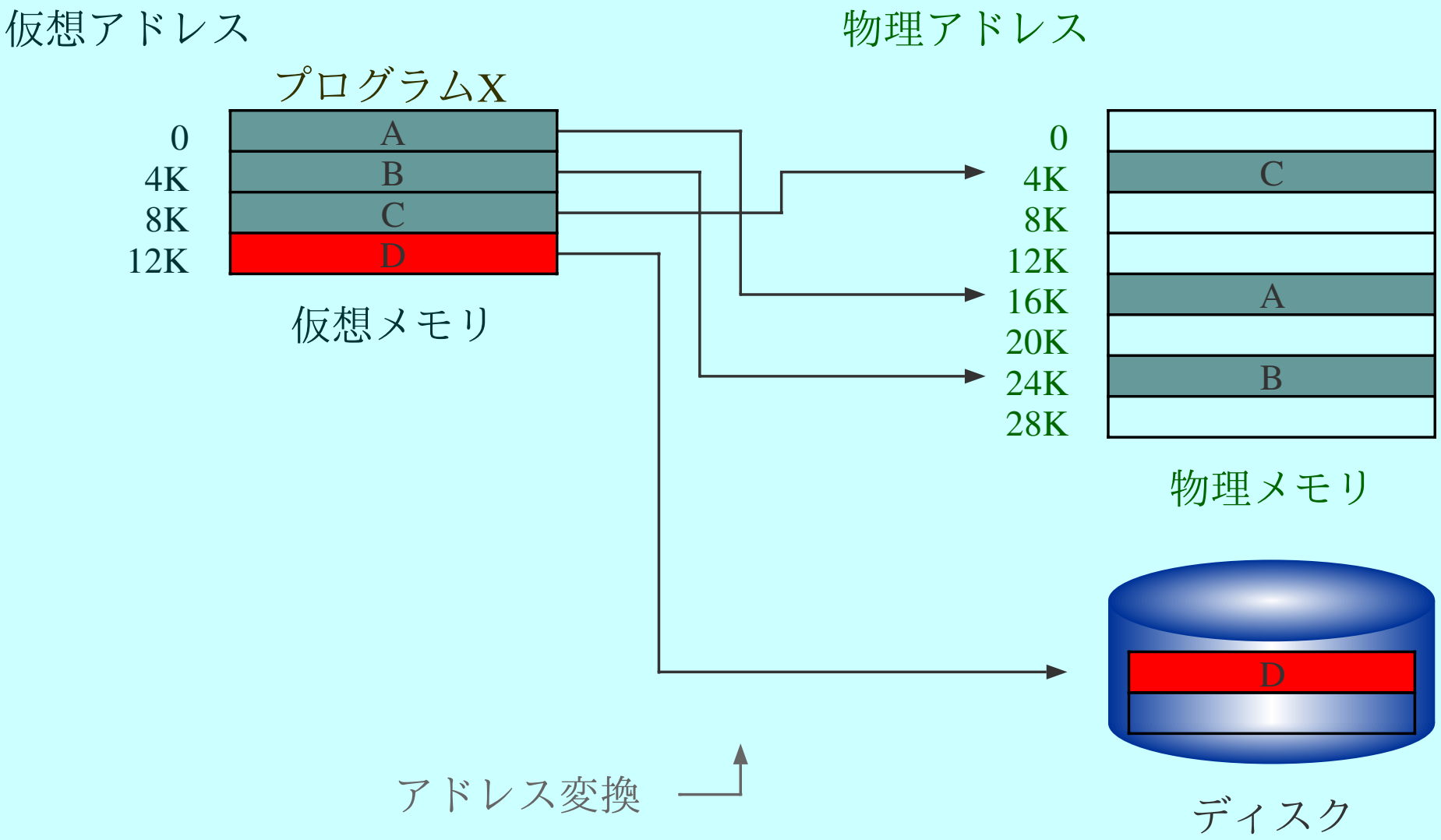

```
addi    x5, x5, -1   # (12c) data cache miss(lw in MEM)
bne     x5, x0, loop # (130) inst cache hit
ret     x1           # (134) inst cache hit
                     # (138) miss, will be canceled
                     # (138)
                     # (138)
                     # (138)
```

Waveform of Data Cache Miss

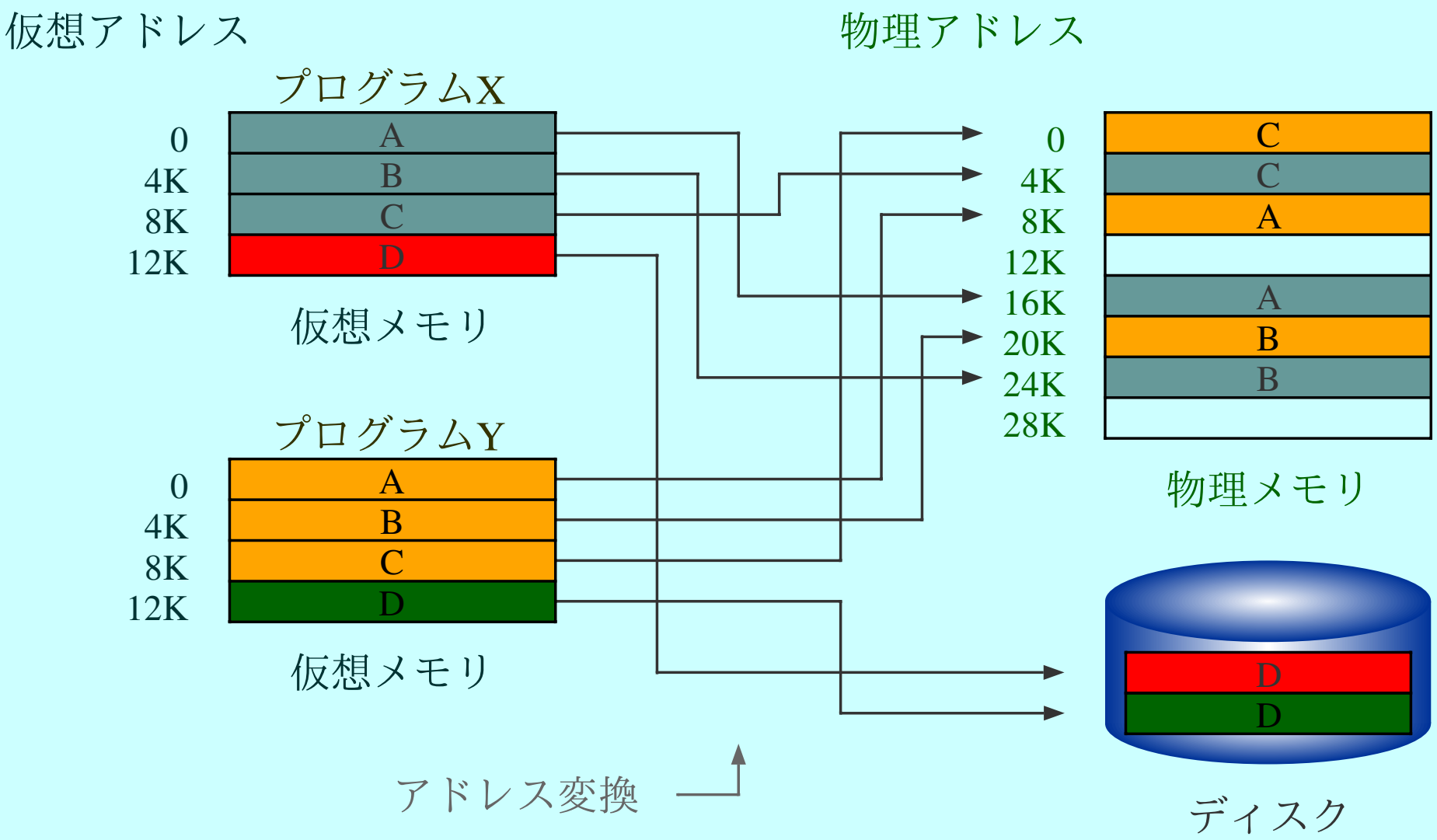



```
                        # (138) miss, will be canceled
sw      x6, 0(x4)       # (00c) inst cache hit
lw      x9, 0(x4)       # (010) inst cache miss
                        # (010)
                        # (010)
                        # (010)
                        # (010)
```

仮想アドレスと物理アドレス



仮想アドレスと物理アドレス



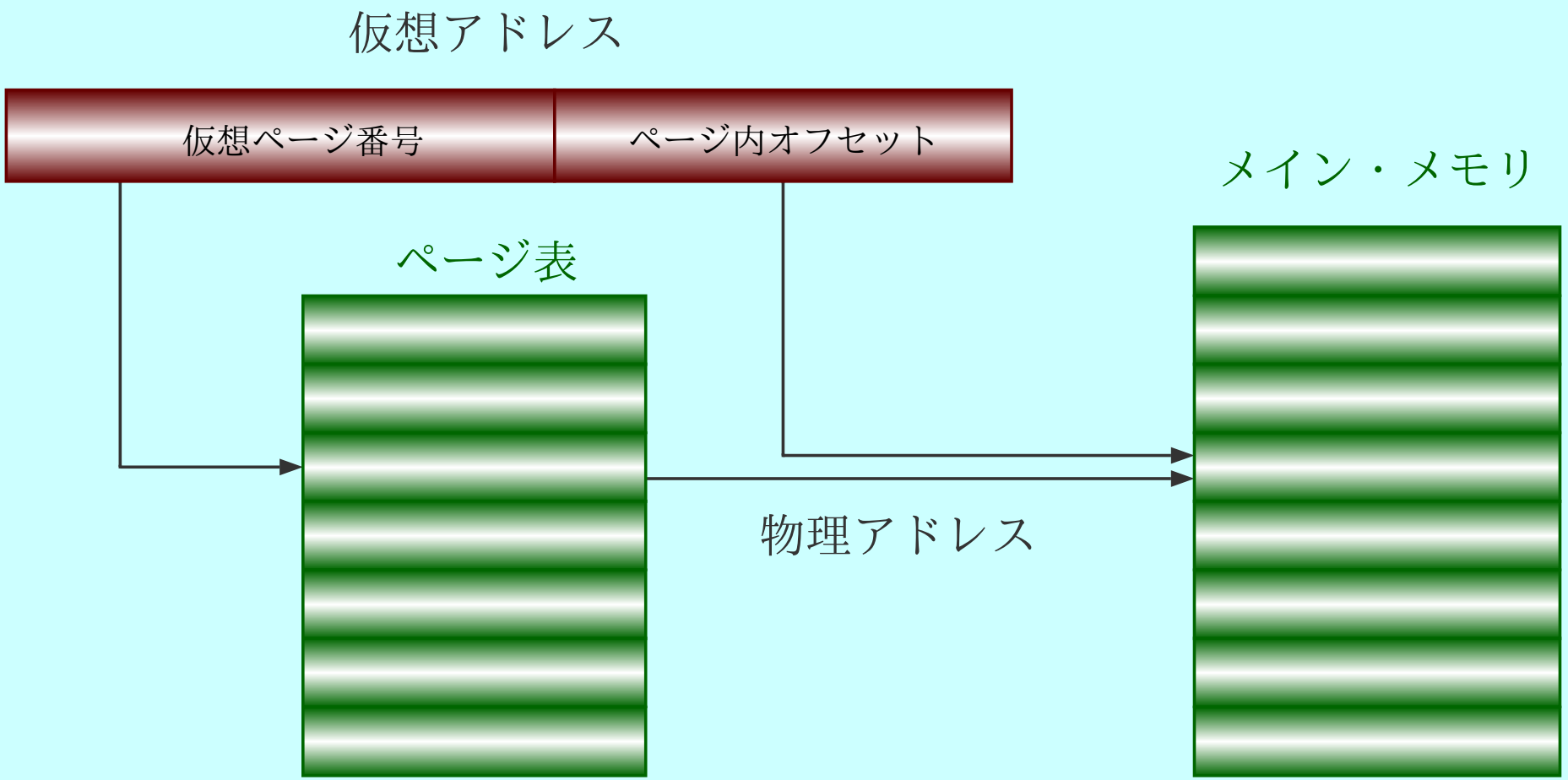
ページとセグメント

- 仮想メモリは二つのクラスに分類される:

1. ページと呼ばれる固定サイズ・ブロックを持つもの
   - ページは 4KB から 64KB に固定される
2. セグメントと呼ばれる可変サイズ・ブロックを持つもの
   - セグメントのサイズは可変である
   - 最大セグメントは 2^{16} バイトから 2^{32} バイトまでに及ぶ
   - 最小セグメントは 1 バイトである

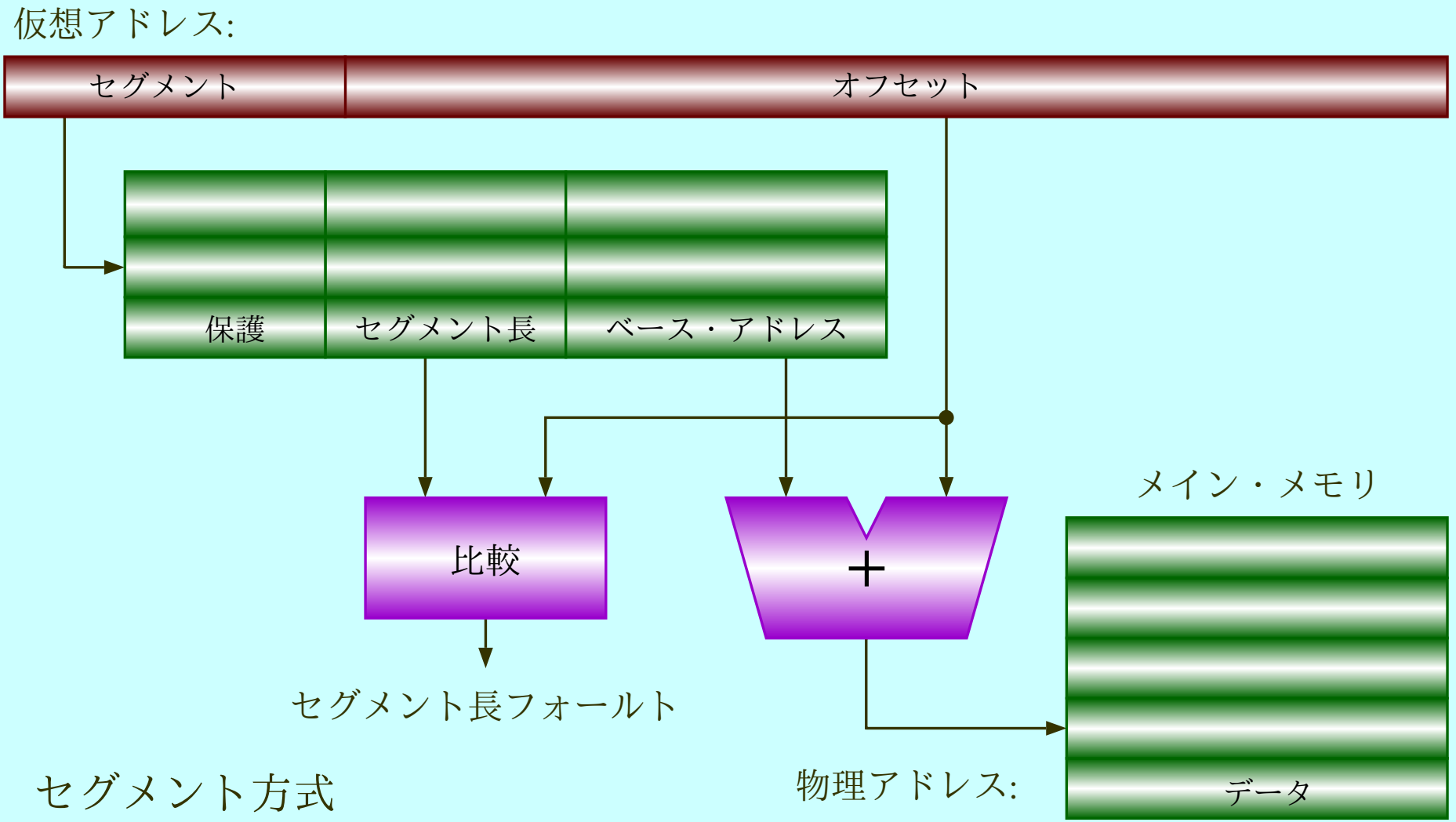
ページ方式　　セグメント方式

仮想アドレス変換 — ページ方式



ページ表を用いた仮想アドレスから物理アドレスへの変換

仮想アドレス変換 — セグメント方式

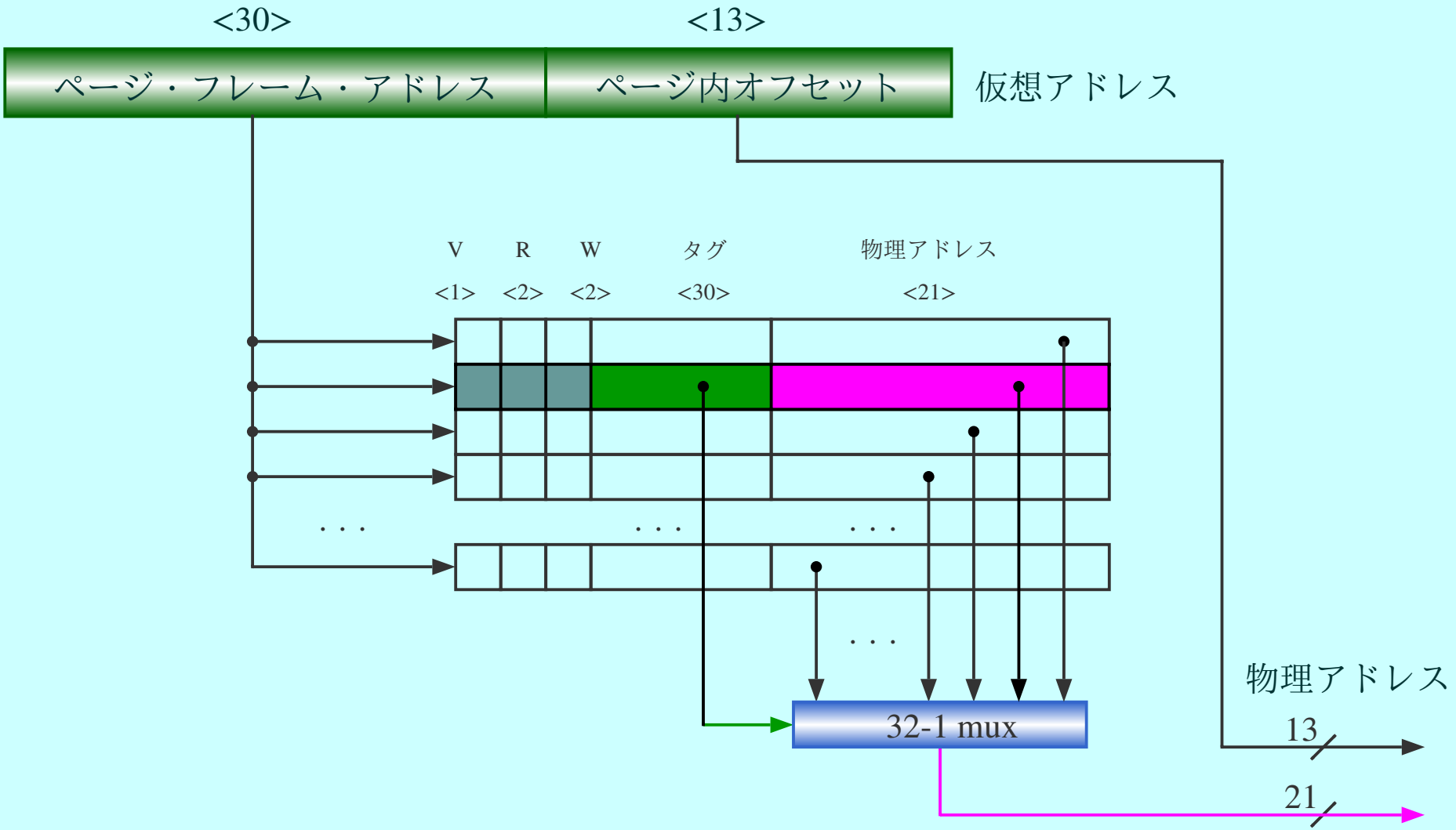


セグメント方式

アドレス変換バッファ — TLB

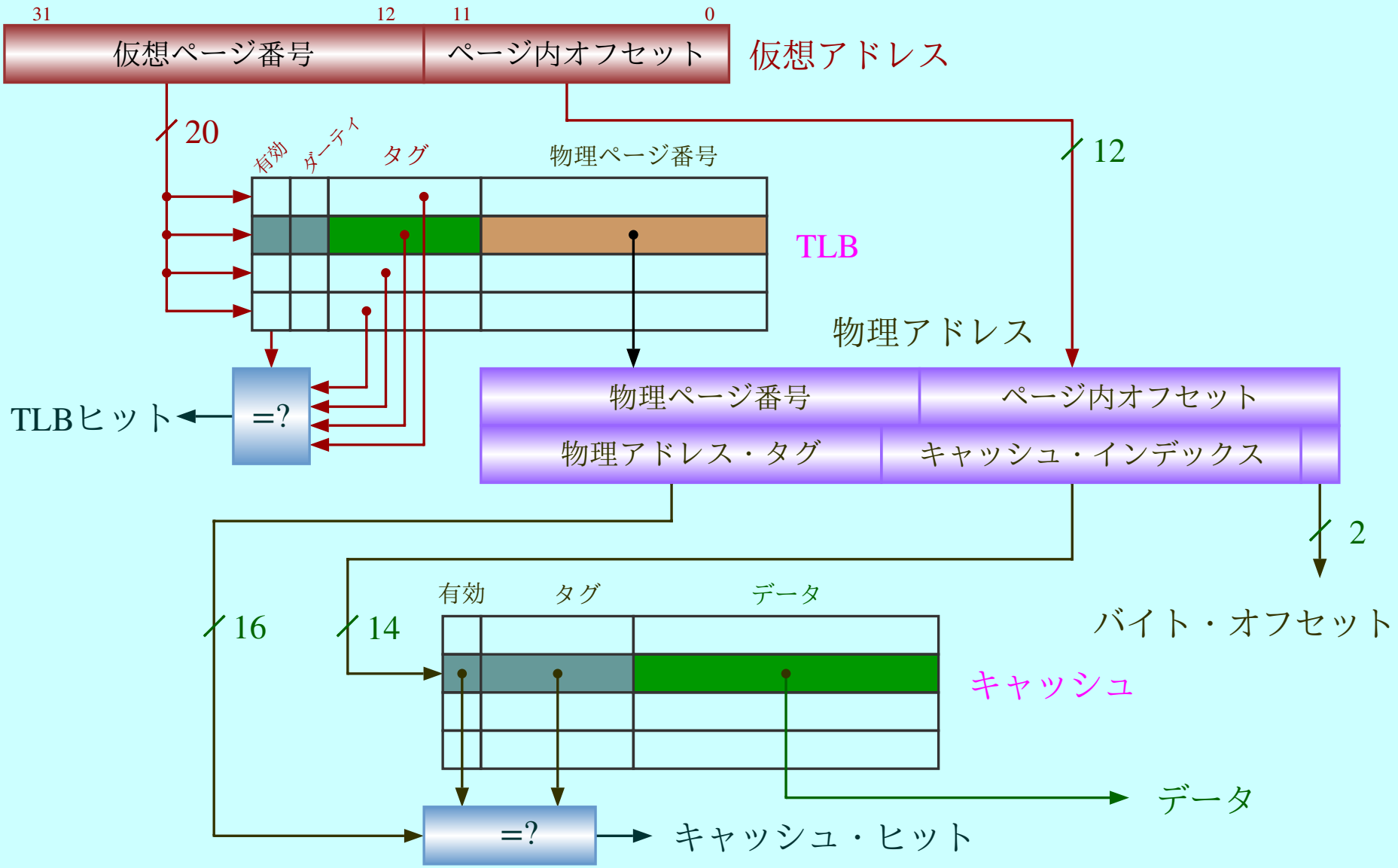
- ページ表はメイン・メモリに置いておける
- これはすべてのメモリ・アクセス命令は必然的に最低 2 回はメモリにアクセスしなければならないことを意味する
  - 物理アドレスを得るためにメモリにアクセスし，そのアドレス上のデータを得るためにまたメモリにアクセスする
- アドレス変換にかかる時間を減らすために，アドレス変換専用のキャッシュを用いる。
  - これはアドレス変換バッファ (TLB — Translation Lookaside Buffer) と呼ばれる

Alpha AXP 21064 TLB



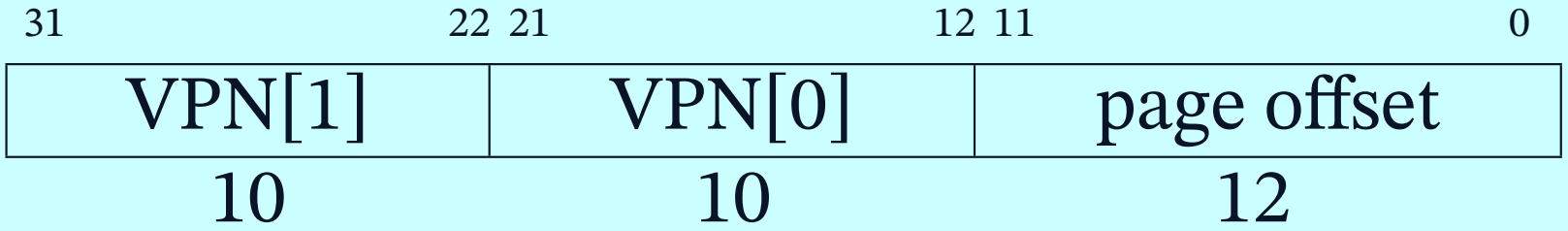
フル・アソシアティブTLB，32エントリ

DECStation 3100 TLB とキャッシュ



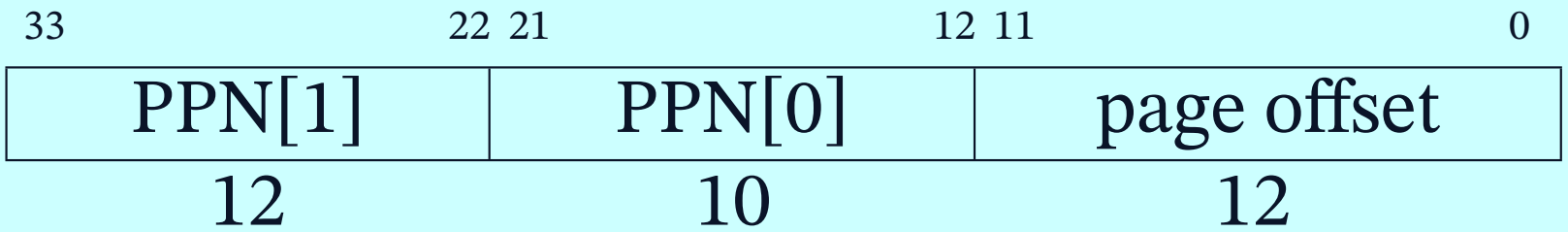
RISC-V Virtual and Physical Addresses

Virtual address



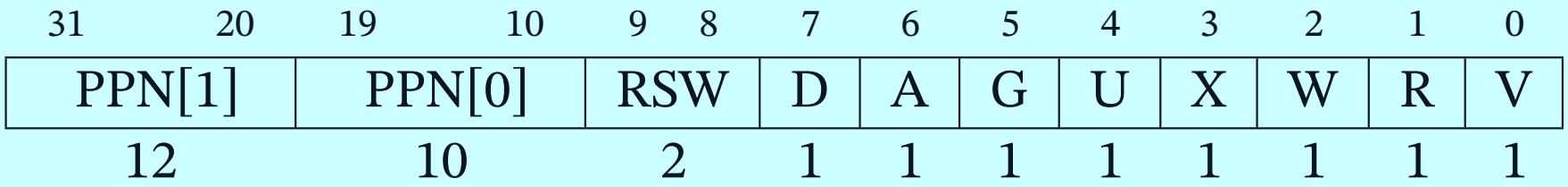
VPN: Virtual Page Number

Physical address



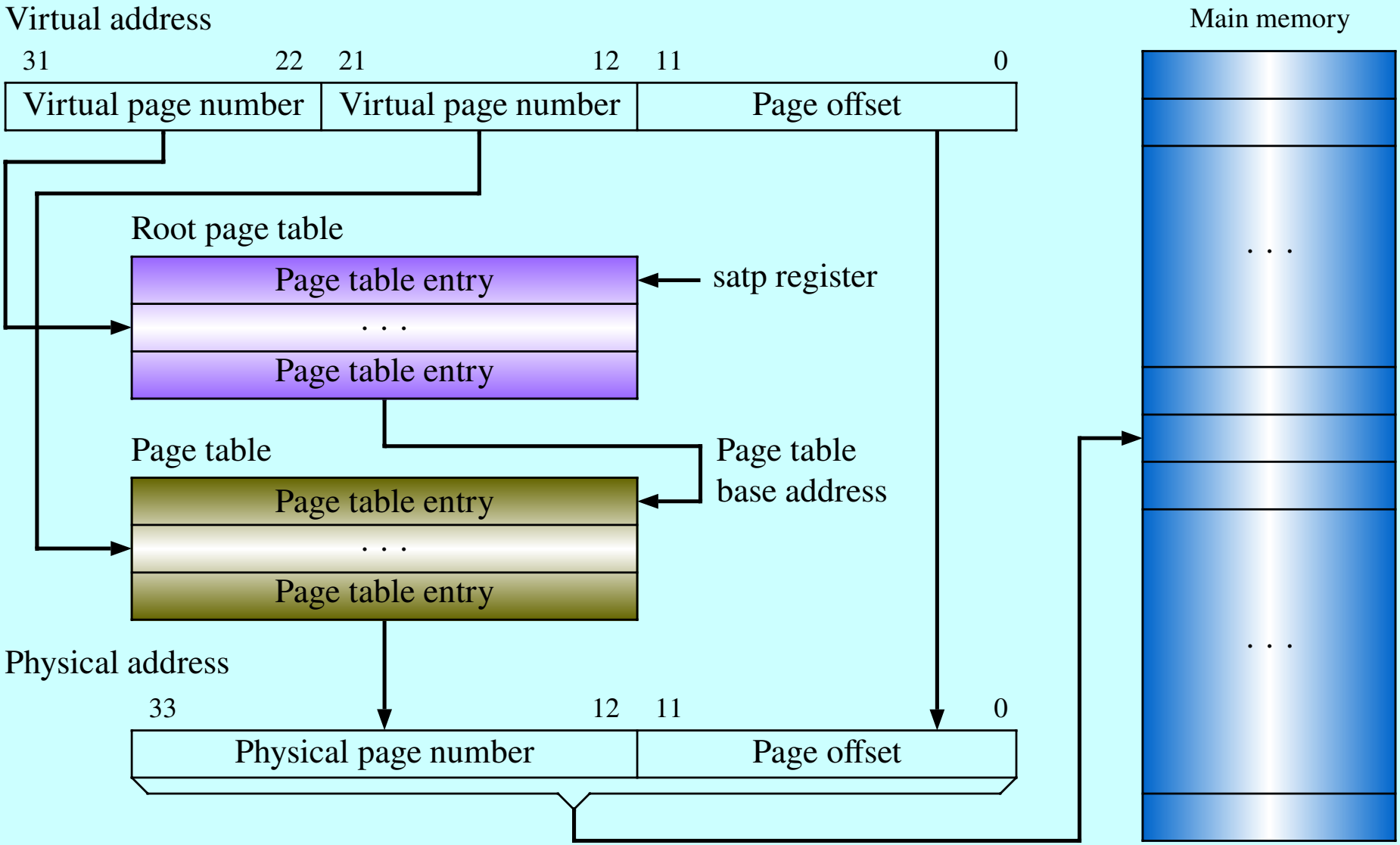
PPN: Physical Page Number

RISC-V Page Table Entry (PTE)



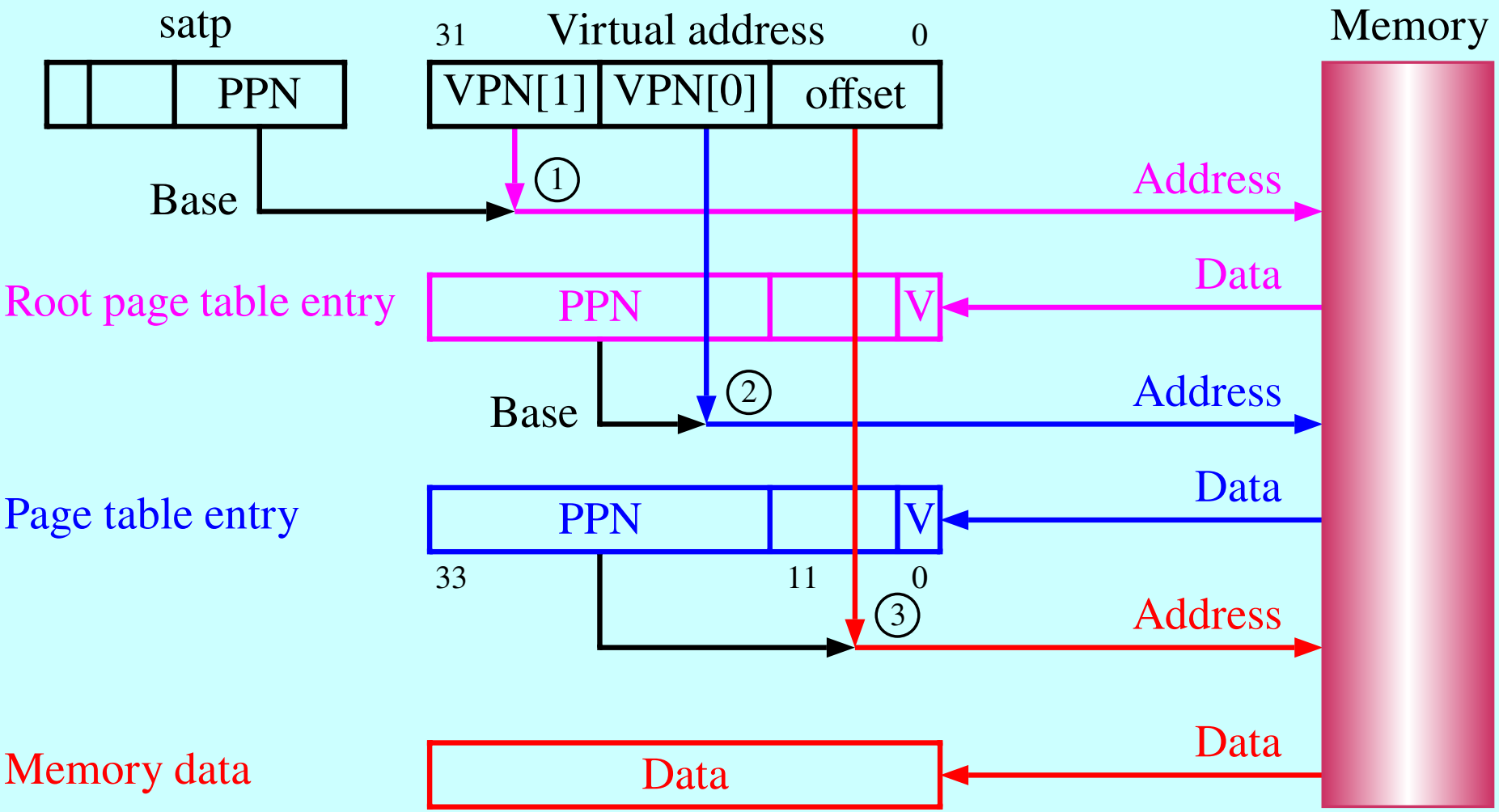
- The physical page number of the root page table is stored in the `satp` register.
- The V bit indicates whether the PTE is valid.
- The permission bits, R, W, and X, indicate whether the page is readable, writable, and executable, respectively. When all three are zero, the PTE is a pointer to the next level of the page table; otherwise, it is a leaf PTE.
- The U bit indicates whether the page is accessible to user mode.
- The G bit designates a *global* mapping.
- The A bit indicates the virtual page has been read, written, or fetched from since the last time the A bit was cleared.
- The D bit indicates the virtual page has been written since the last time the D bit was cleared.
- The RSW field is reserved for use by supervisor software; the implementation shall ignore this field.

RISC-V Address Translation

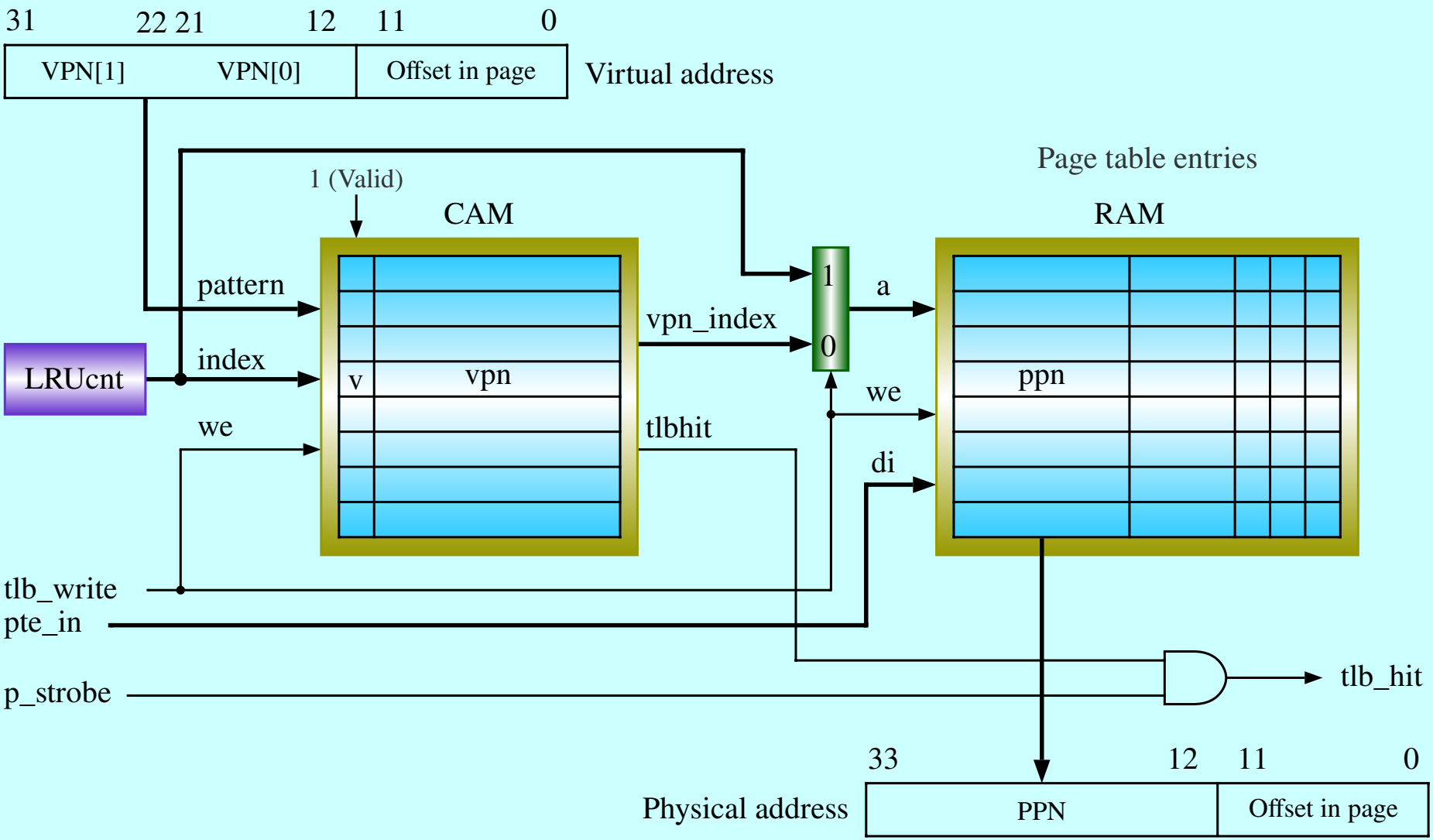


RISC-V RV32 Memory Management

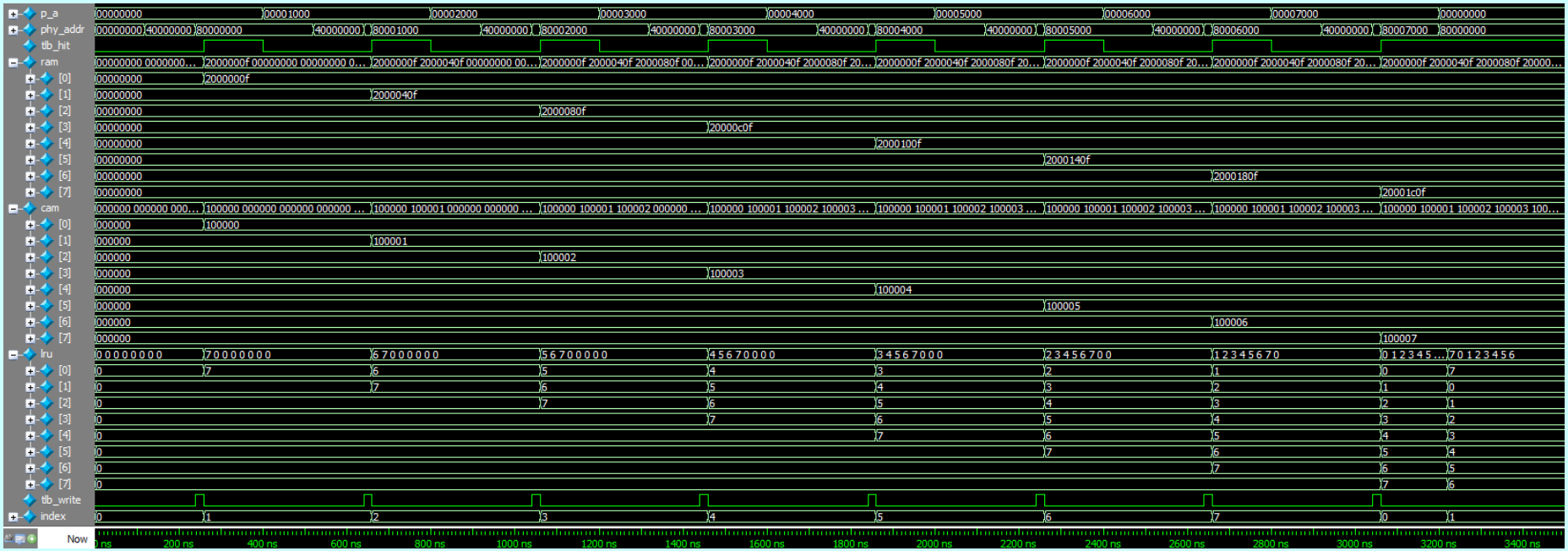
Supervisor address translation and protection (satp) register



Translation Lookaside Buffer Design



Waveform of TLB Miss

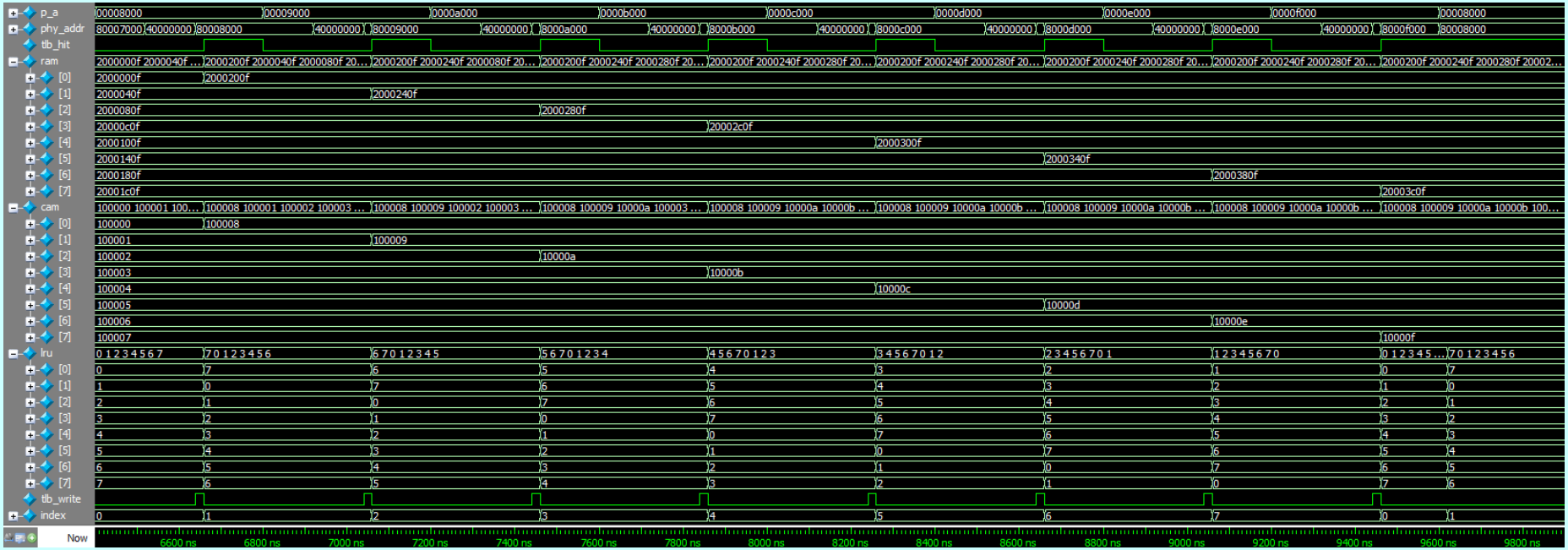


Virtual address 0x00000000 is mapped to physical address 0x80000000
Virtual address 0x00001000 is mapped to physical address 0x80001000
Virtual address 0x00002000 is mapped to physical address 0x80002000
Virtual address 0x00003000 is mapped to physical address 0x80003000
Virtual address 0x00004000 is mapped to physical address 0x80004000
Virtual address 0x00005000 is mapped to physical address 0x80005000
Virtual address 0x00006000 is mapped to physical address 0x80006000
Virtual address 0x00007000 is mapped to physical address 0x80007000

Waveform of TLB Hit

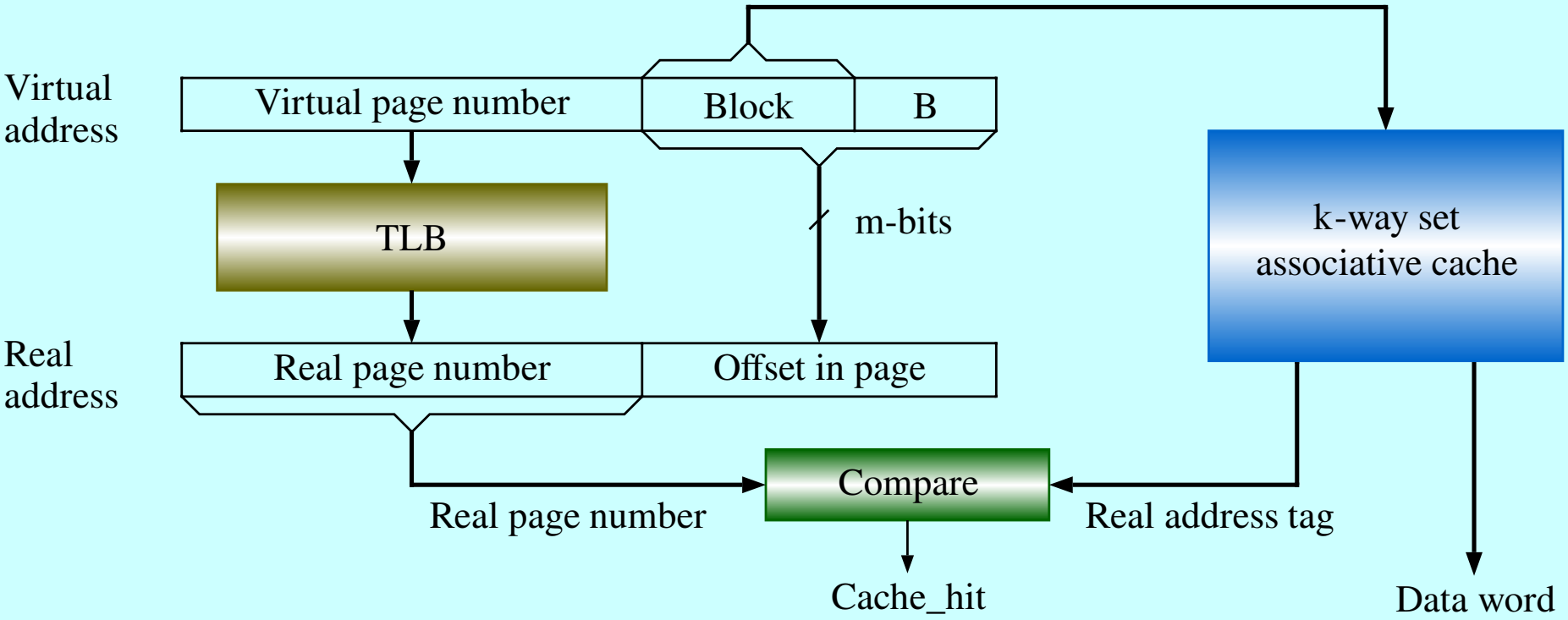
Virtual address 0x00000000 is mapped to physical address 0x80000000
Virtual address 0x00001000 is mapped to physical address 0x80001000
Virtual address 0x00002000 is mapped to physical address 0x80002000
Virtual address 0x00003000 is mapped to physical address 0x80003000
Virtual address 0x00004000 is mapped to physical address 0x80004000
Virtual address 0x00005000 is mapped to physical address 0x80005000
Virtual address 0x00006000 is mapped to physical address 0x80006000
Virtual address 0x00007000 is mapped to physical address 0x80007000

Waveform of TLB Miss



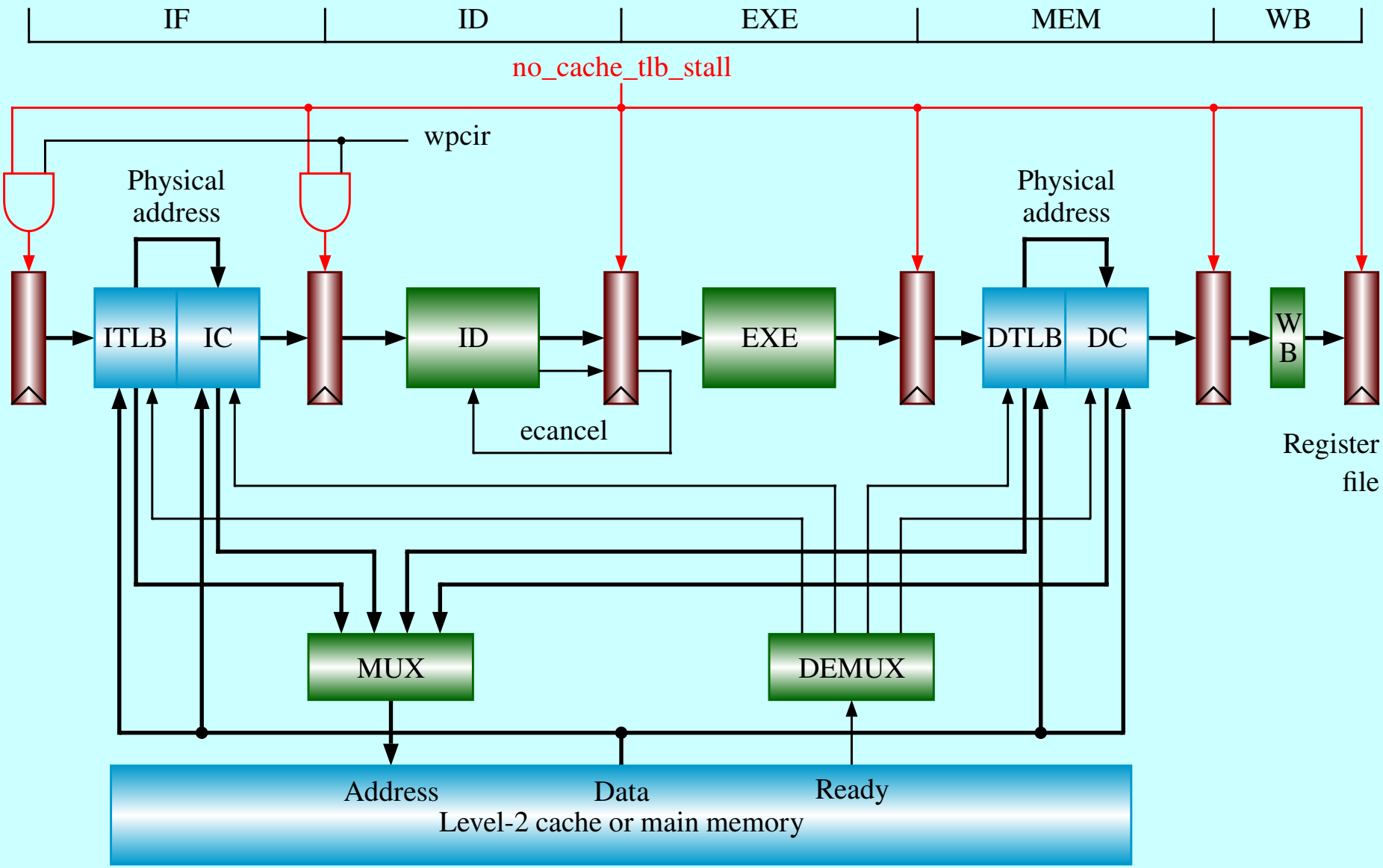
Virtual address 0x00008000 is mapped to physical address 0x80008000
Virtual address 0x00009000 is mapped to physical address 0x80009000
Virtual address 0x0000a000 is mapped to physical address 0x8000a000
Virtual address 0x0000b000 is mapped to physical address 0x8000b000
Virtual address 0x0000c000 is mapped to physical address 0x8000c000
Virtual address 0x0000d000 is mapped to physical address 0x8000d000
Virtual address 0x0000e000 is mapped to physical address 0x8000e000
Virtual address 0x0000f000 is mapped to physical address 0x8000f000

Cache and TLB Accesses in Parallel

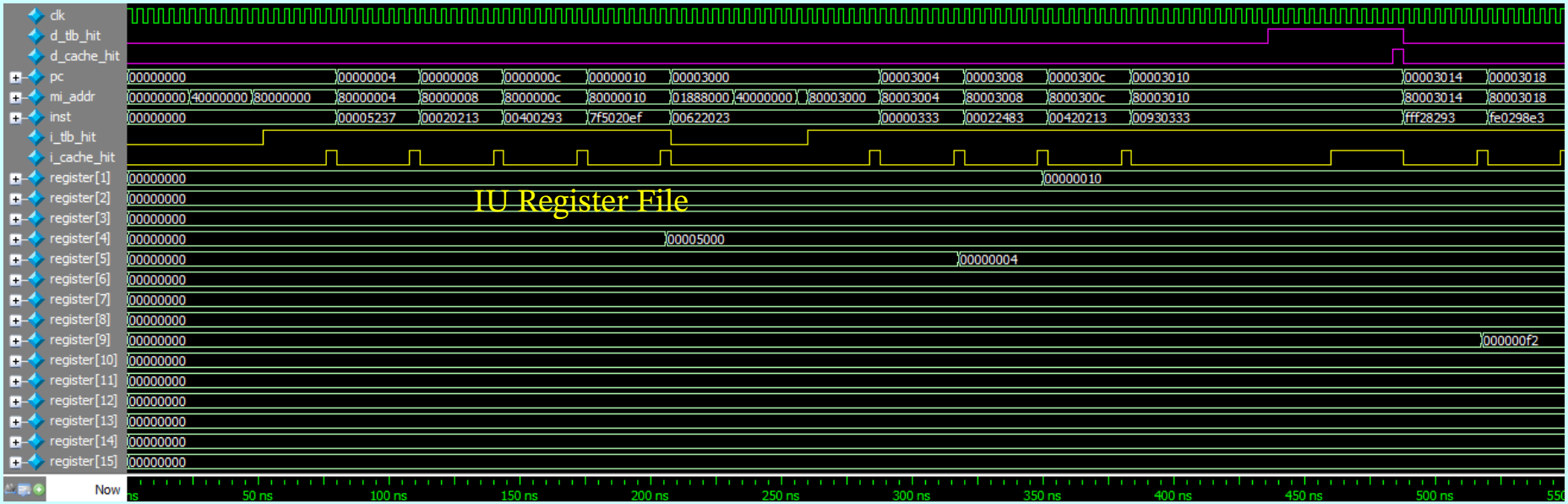


Cache and TLB access in parallel: The cache size should not be larger than $2^m k$ bytes where k is the number of ways in a set and 2^m is the page size.

Pipelined CPU + Caches + TLBs

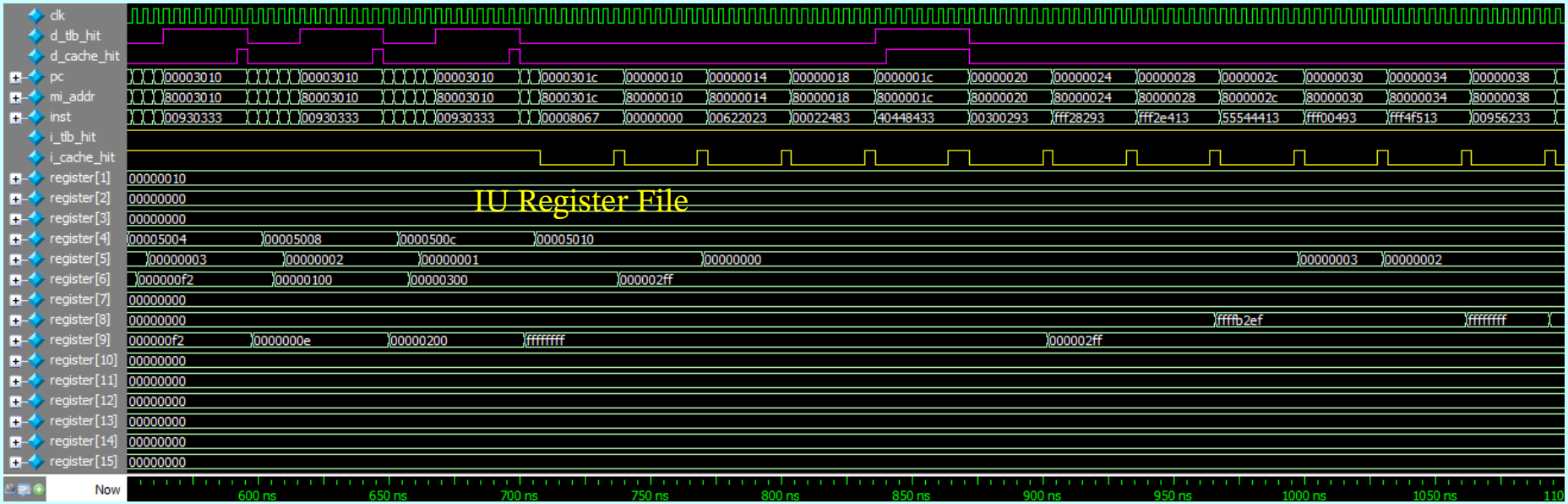


Pipelined CPU + Caches + TLBs




```
(0000) main:   lui    x4, %hi(array1)     # (0000) x4 <- array1: 0x5000 virtual address
(0004)         addi   x4, x4, %lo(array1) # (0004) x4 <- array1: 0x5000 virtual address
(0008)         addi   x5, x0,  4          # (0008) x5 <- 4
(000c) call:   jal    x1, sum             # (000c) x1 <- 0x10 (return address), call sum
(0010)         sw     x6, 0(x4)           # (0010) memory[x4+0] <- x6, canceled
(3000) sum:    add    x6, x0, x0          # (3000) x6 <- 0 (subroutine entry)
(3004) loop:   lw     x9, 0(x4)           # (3004) x9 <- memory[x4+0]
(3008)         addi   x4, x4,  4          # (3008) x4 <- x4 + 4 (address+4)
(300c)         add    x6, x6, x9          # (300c) x6 <- x6 + x9 (sum)
(3010)         addi   x5, x5, -1          # (3010) x5 <- x5 - 1 (counter--)
(3014)         bne    x5, x0, loop        # (3014) if x5 != 0, goto loop
(3018)         ret    x1                  # (3018) return from subroutine 100 1101 00
```

Pipelined CPU + Caches + TLBs

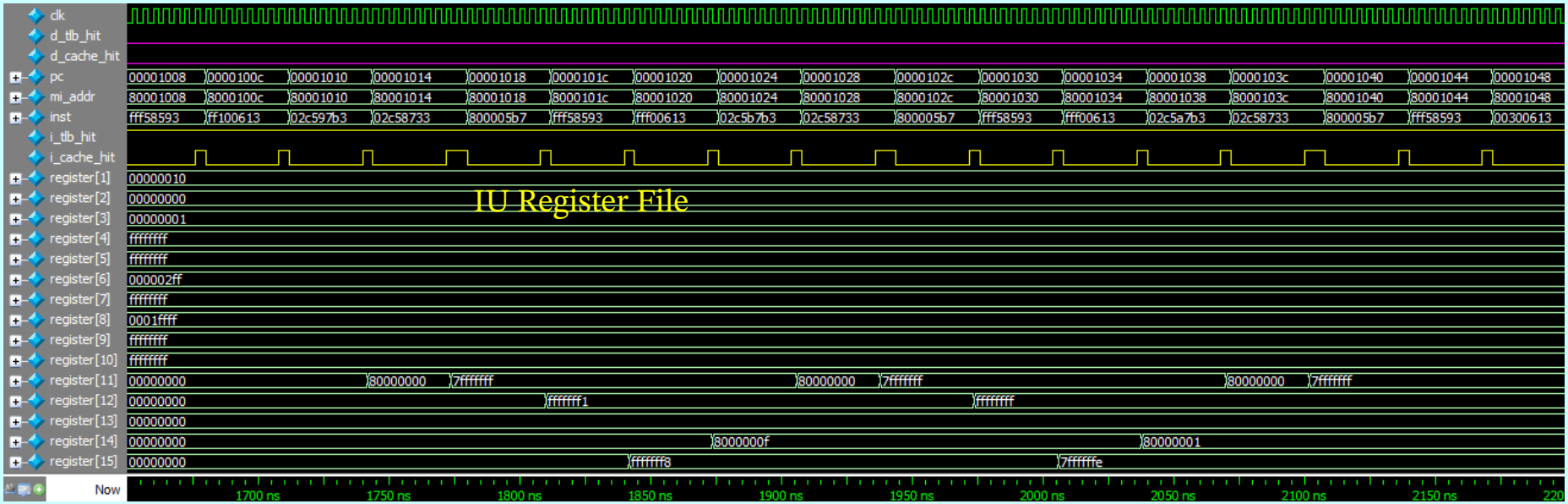



```
(3018)          ret     x1              # (3018) return from subroutine 100 1101 00
(301c)                                  # (301c) canceled
(0010)          sw      x6, 0(x4)       # (0010) memory[x4+0] <- x6
(0014)          lw      x9, 0(x4)       # (0014) x6 <- memory[x4+0]
(0018)          sub     x8, x9, x4      # (0018) x8 <- x9 - x4
(001c)          addi    x5, x0,  3      # (001c) x5 <- 3
(0020) loop2:   addi    x5, x5, -1      # (0020) x5 <- x5 - 1
(0024)          ori     x8, x5, 0xfff   # (0024) x8 <- x5 | 0xffffffff = 0xffffffff
(0028)          xori    x8, x8, 0x555   # (0028) x8 <- x8 ^ 0x00000555 = 0xfffffaaa
(002c)          addi    x9, x0, -1      # (002c) x9 <- 0xffffffff
(0030)          andi    x10,x9, 0xfff   # (0030) x10<- x9 & 0xffffffff = 0xffffffff
(0034)          or      x4, x10, x9     # (0034) x4 <- x10 | x9 = 0xffffffff
```

Pipelined CPU + Caches + TLBs

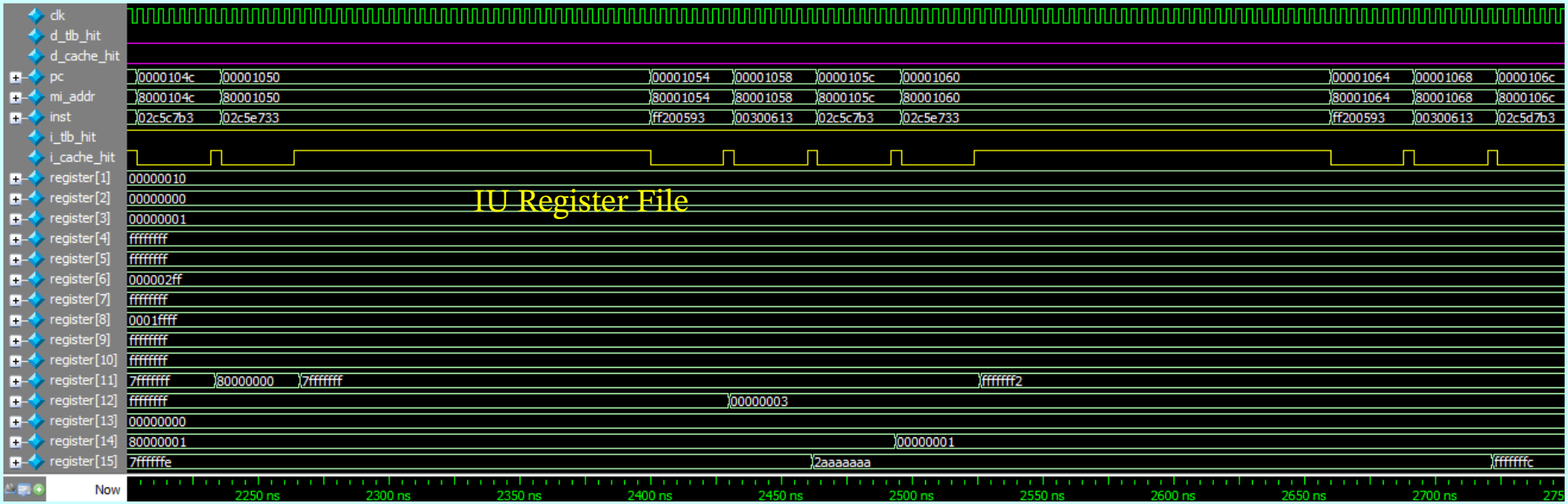

```
(003c)          and     x7, x10, x4         # (003c) x7 <- x10 & x4 = 0xffffffff
(0040)          beq     x5, x0, shift       # (0040) if x5 = 0, goto shift
(0044)          jal     x0, loop2           # (0044) jump loop2
(0048) shift:   addi    x5, x0, -1          # (0048) x5 <- 0xffffffff
(004c)          slli    x8, x5, 15          # (004c) x8 <- 0xffffffff <<  15 = 0xffff8000
(0050)          slli    x8, x8, 16          # (0050) x8 <- 0xffff8000 <<  16 = 0x80000000
(0054)          srai    x8, x8, 16          # (0054) x8 <- 0x80000000 >>> 16 = 0xffff8000
(0058)          srli    x8, x8, 15          # (0058) x8 <- 0xffff8000 >>  15 = 0x0001ffff
(005c)          slt     x3, x4, x6          # (005c) x3 <- 0xffffffff < 0x000002ff = 1
(0060)          jal     x0, s_x_s           # (0060) jump s_x_s
(1000) s_x_s:   li      a1, 0x7fffffff      # (1000) a               H = 0xfffffff8
(1004)          li      a1, 0x7fffffff      # (1004) a               H = 0xfffffff8
```

Pipelined CPU + Caches + TLBs




```
(1008)          li      a2, -15             # (1008) b                  L = 0x8000000f
(100c)          mulh    a5,  a1, a2         # (100c) product high
(1010)          mul     a4,  a1, a2         # (1010) product low, fused with mulh
(1014) u_x_u:   li      a1, 0x7fffffff      # (1014) a                  H = 0x7ffffffe
(1018) u_x_u:   li      a1, 0x7fffffff      # (1018) a                  H = 0x7ffffffe
(101c)          li      a2, -1              # (101c) b                  L = 0x80000001
(1020)          mulhu   a5,  a1, a2         # (1020) product high
(1024)          mul     a4,  a1, a2         # (1024) product low, fused with mulhu
(1028) s_x_u:   li      a1, 0x7fffffff      # (1028) a                  H = 0x7ffffffe
(102c)          li      a1, 0x7fffffff      # (102c) a                  H = 0x7ffffffe
(1030)          li      a2, -1              # (1030) b                  L = 0x80000001
(1034)          mulhsu  a5,  a1, a2         # (1034) product high
```

Pipelined CPU + Caches + TLBs

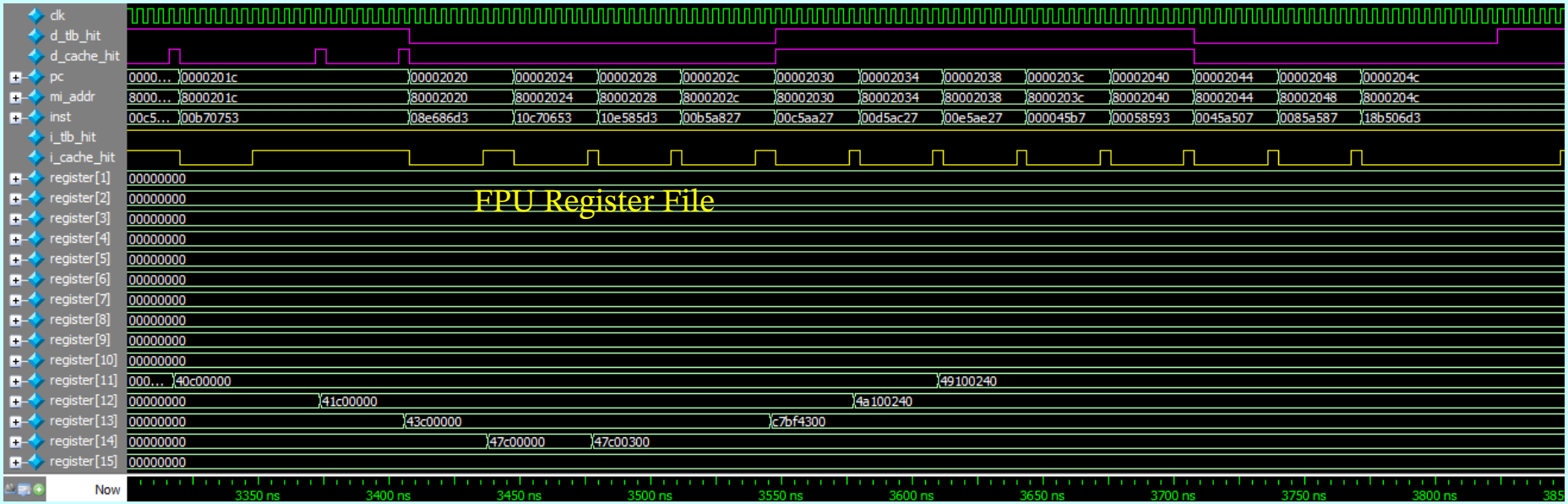



```
(1040)          li      a1, 0x7fffffff      # (1040) a               Q = 0x2aaaaaaa
(1044)          li      a2,   3             # (1044) b               R = 0x00000001
(1048)          div     a5,  a1, a2         # (1048) div signed
(104c)          rem     a4,  a1, a2         # (104c) rem signed, fused with div
(1050) sign1:   li      a1, 0xfffffff2      # (1050) a               Q = 0xfffffffc
(1054)          li      a2,   3             # (1054) b               R = 0xfffffffe
(1058)          div     a5,  a1, a2         # (1058) div signed
(105c)          rem     a4,  a1, a2         # (105c) rem signed, fused with div
(1060) unsign:  li      a1, 0xfffffff2      # (1060) a               Q = 0x55555550
(1064)          li      a2,   3             # (1064) b               R = 0x00000002
(1068)          divu    a5,  a1, a2         # (1068) div unsigned
(106c)          remu    a4,  a1, a2         # (106c) rem unsigned, fused with divu
```

Pipelined CPU + Caches + TLBs

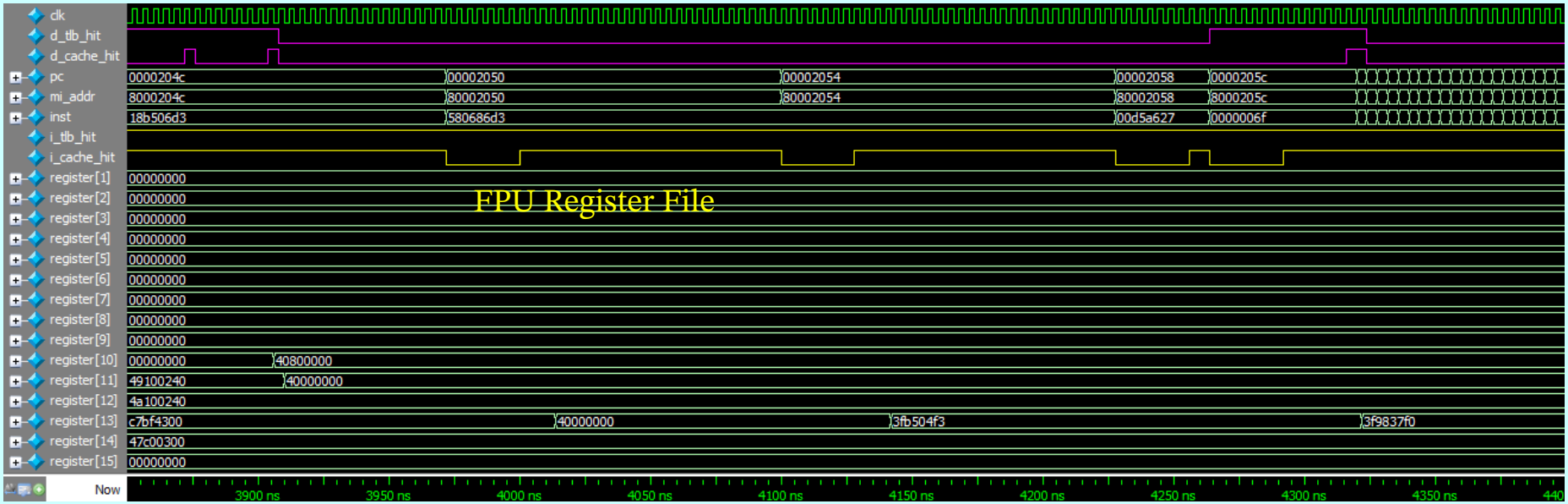

```
(1070)          jal     x0, fpu             # (1070) jump fpu
(1074)                                      # (1074) canceled
(2000) fpu:     lui     a1, %hi(array2)     # (2000) a1 <- array2: 0x00006000 virtual address
(2004)          addi    a1, a1, %lo(array2) # (2004) a1 <- array2: 0x00006000 virtual address
(2008)          flw     fa1,   0(a1)        # (2008) 0x40c00000
(200c)          flw     fa2,   4(a1)        # (200c) 0x41c00000
(2010)          flw     fa3,   8(a1)        # (2010) 0x43c00000
(2014)          flw     fa4,  12(a1)        # (2014) 0x47c00000
(2018)          fadd.s  fa4, fa4, fa1       # (2018) (47c00000) + (40c00000) = (47c00300)
(201c)          fsub.s  fa3, fa3, fa4       # (201c) (43c00000) - (47c00300) = (c7bf4300)
(2020)          fmul.s  fa2, fa4, fa2       # (2020) (47c00300) * (41c00000) = (4a100240)
(2024)          fmul.s  fa1, fa1, fa4       # (2024) (40c00000) * (47c00300) = (49100240)
```

Pipelined CPU + Caches + TLBs



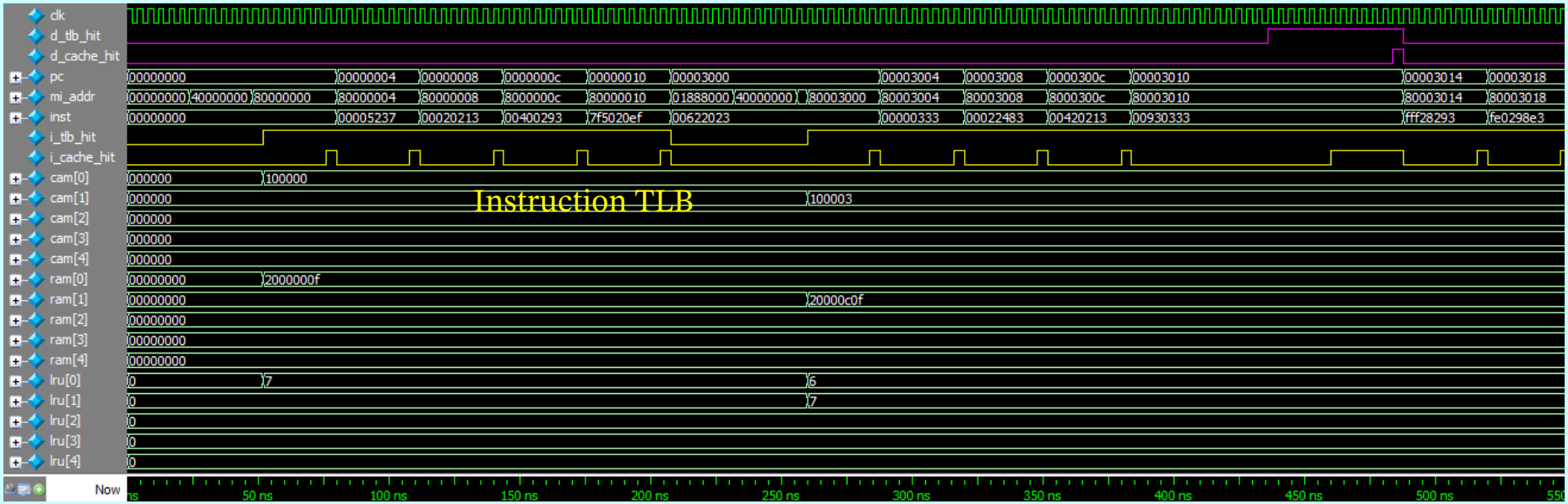

```
(201c)          fsub.s  fa3, fa3, fa4       # (201c) (43c00000) - (47c00300) = (c7bf4300)
(2020)          fmul.s  fa2, fa4, fa2       # (2020) (47c00300) * (41c00000) = (4a100240)
(2024)          fmul.s  fa1, fa1, fa4       # (2024) (40c00000) * (47c00300) = (49100240)
(2028)          fsw     fa1,  16(a1)        # (2028)
(202c)          fsw     fa2,  20(a1)        # (202c)
(2030)          fsw     fa3,  24(a1)        # (2030)
(2034)          fsw     fa4,  28(a1)        # (2034)
(2038)          lui     a1, %hi(array0)     # (2038) a1 <- array0: 0x00004000 virtual address
(203c)          addi    a1, a1, %lo(array0) # (203c) a1 <- array0: 0x00004000 virtual address
(2040)          flw     fa0,   4(a1)        # (2040)  40800000
(2044)          flw     fa1,   8(a1)        # (2044)  40000000
(2048)          fdiv.s  fa3, fa0, fa1       # (2048) (40800000) / (40000000) = (40000000)
```

Pipelined CPU + Caches + TLBs



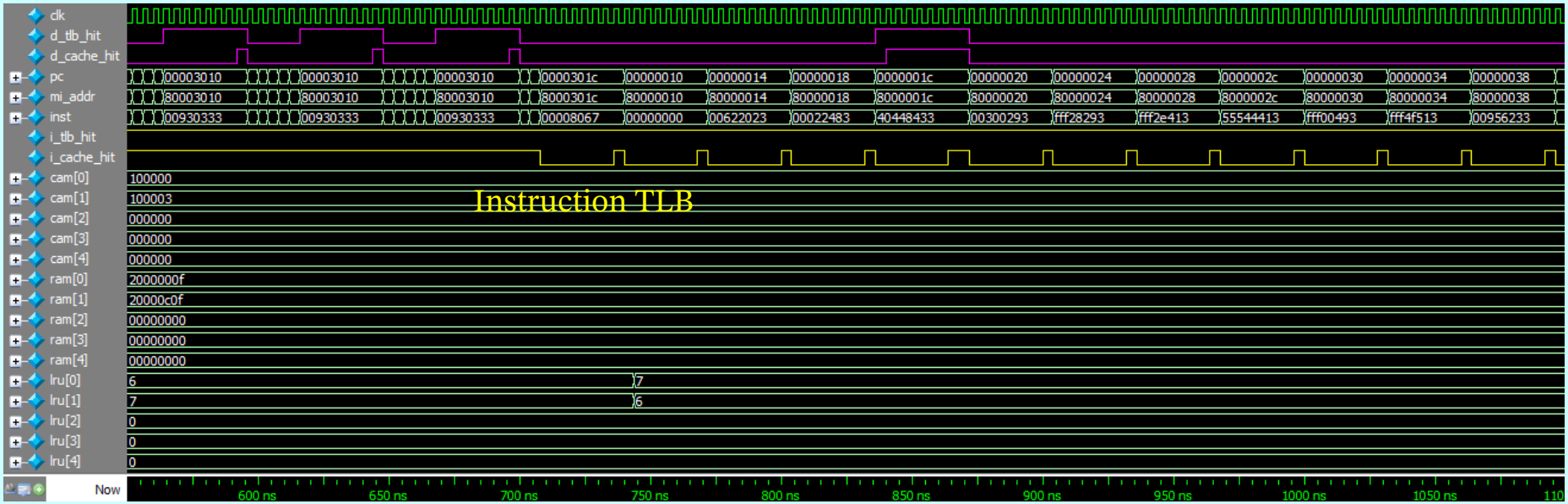

```
(204c)          fsqrt.s fa3, fa3          # (204c) (40000000)      sqrt    = (3fb504f3)
(2050)          fsqrt.s fa3, fa3          # (2050) (3fb504f3)      sqrt    = (3f9837f0)
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
```

Waveform of Instruction TLB



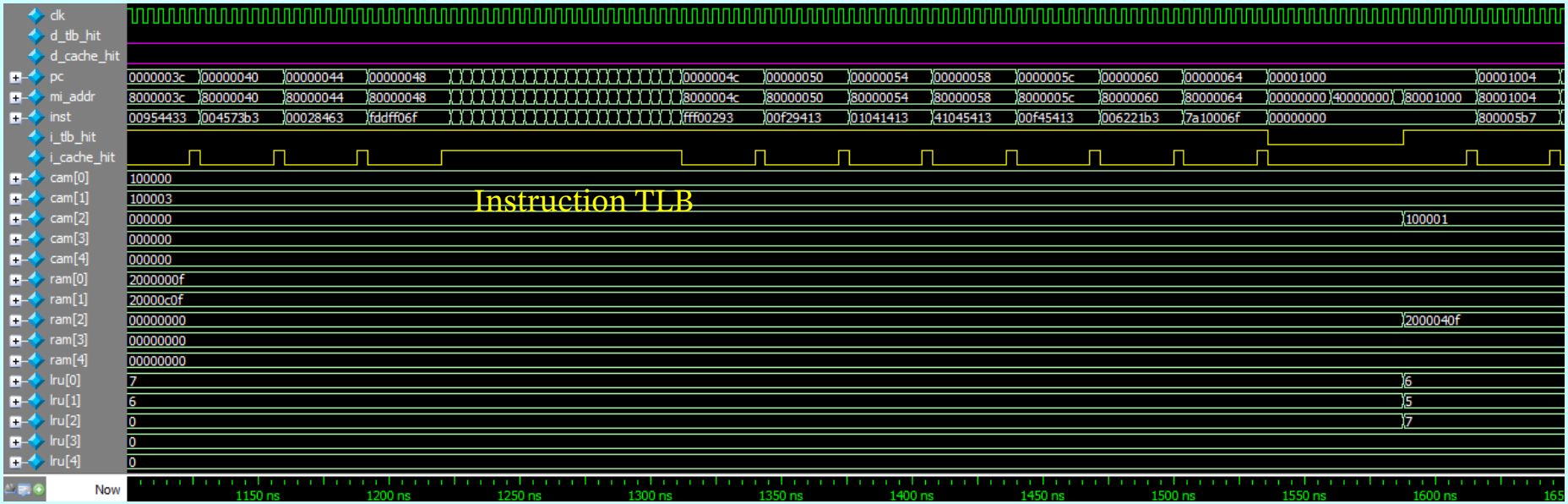

```
(0000) main:   lui     x4, %hi(array1)     # (0000) x4 <- array1: 0x5000 virtual address
(0004)         addi    x4, x4, %lo(array1) # (0004) x4 <- array1: 0x5000 virtual address
(0008)         addi    x5, x0,  4          # (0008) x5 <- 4
(000c) call:   jal     x1, sum             # (000c) x1 <- 0x10 (return address), call sum
(0010)         sw      x6, 0(x4)           # (0010) memory[x4+0] <- x6, canceled
(3000) sum:    add     x6, x0, x0          # (3000) x6 <- 0 (subroutine entry)
(3004) loop:   lw      x9, 0(x4)           # (3004) x9 <- memory[x4+0]
(3008)         addi    x4, x4,  4          # (3008) x4 <- x4 + 4 (address+4)
(300c)         add     x6, x6, x9          # (300c) x6 <- x6 + x9 (sum)
(3010)         addi    x5, x5, -1          # (3010) x5 <- x5 - 1 (counter--)
(3014)         bne     x5, x0, loop        # (3014) if x5 != 0, goto loop
(3018)         ret     x1                  # (3018) return from subroutine 100 1101 00
```

Waveform of Instruction TLB



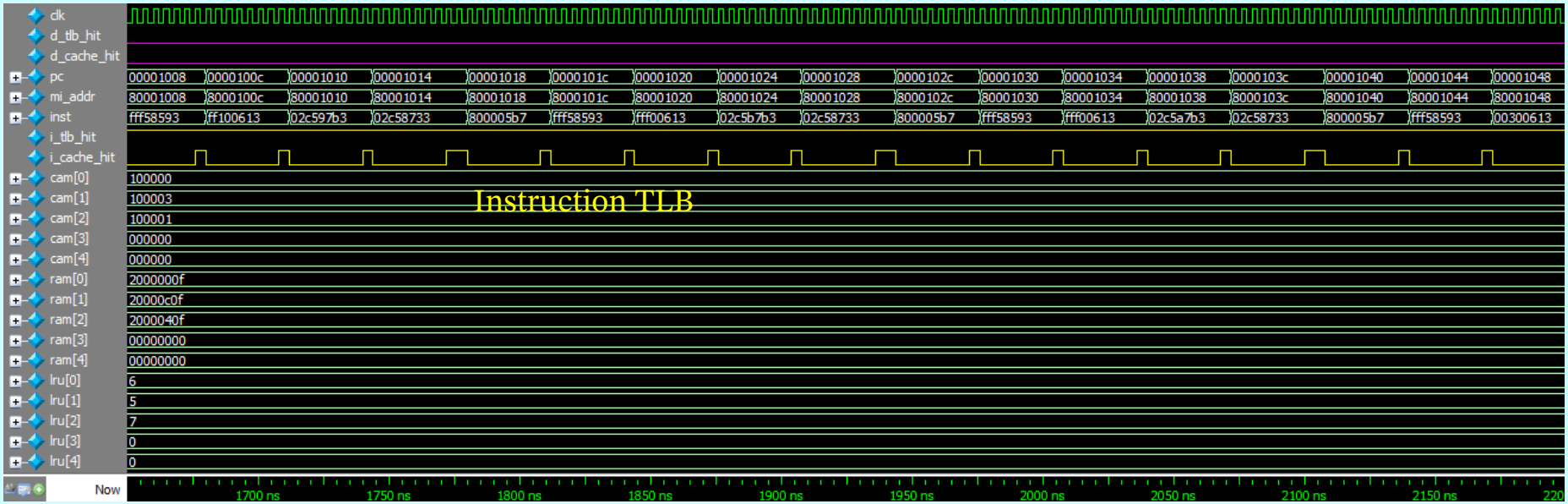

```
(3018)          ret     x1              # (3018) return from subroutine 100 1101 00
(301c)                                  # (301c) canceled
(0010)          sw      x6, 0(x4)       # (0010) memory[x4+0] <- x6
(0014)          lw      x9, 0(x4)       # (0014) x6 <- memory[x4+0]
(0018)          sub     x8, x9, x4      # (0018) x8 <- x9 - x4
(001c)          addi    x5, x0,  3      # (001c) x5 <- 3
(0020) loop2:   addi    x5, x5, -1      # (0020) x5 <- x5 - 1
(0024)          ori     x8, x5, 0xfff   # (0024) x8 <- x5 | 0xffffffff = 0xffffffff
(0028)          xori    x8, x8, 0x555   # (0028) x8 <- x8 ^ 0x00000555 = 0xfffffaaa
(002c)          addi    x9, x0, -1      # (002c) x9 <- 0xffffffff
(0030)          andi    x10,x9, 0xfff   # (0030) x10<- x9 & 0xffffffff = 0xffffffff
(0034)          or      x4, x10, x9     # (0034) x4 <- x10 | x9 = 0xffffffff
```

Waveform of Instruction TLB



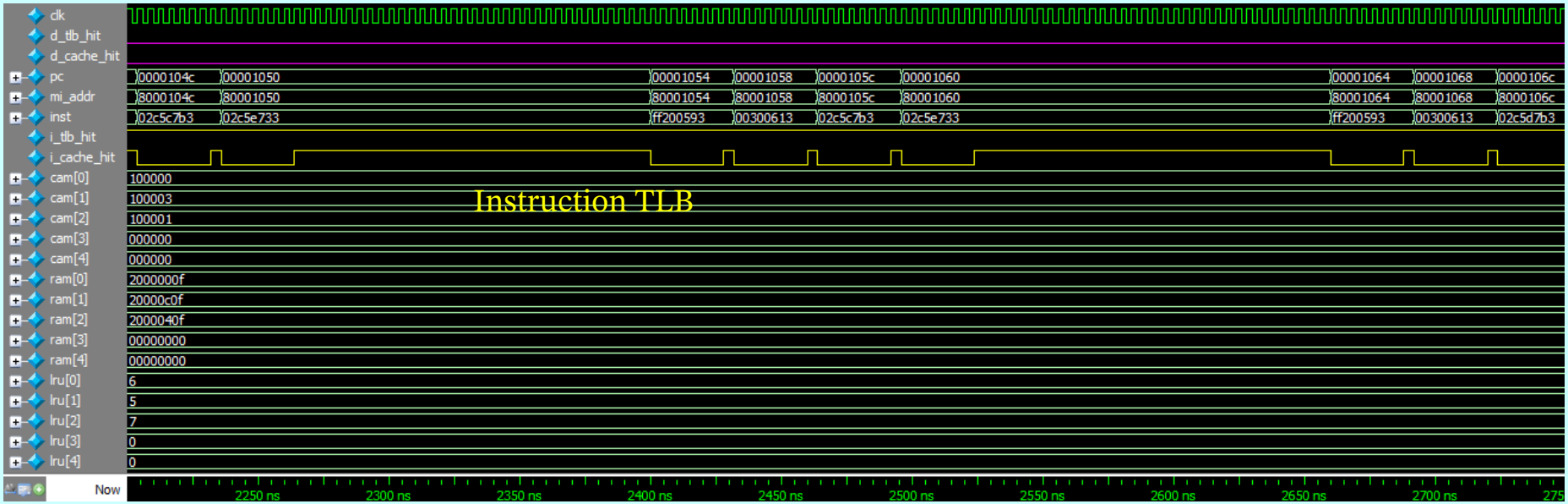

```
(003c)          and     x7, x10, x4         # (003c) x7 <- x10 & x4 = 0xffffffff
(0040)          beq     x5, x0, shift       # (0040) if x5 = 0, goto shift
(0044)          jal     x0, loop2           # (0044) jump loop2
(0048) shift:   addi    x5, x0, -1          # (0048) x5 <- 0xffffffff
(004c)          slli    x8, x5, 15          # (004c) x8 <- 0xffffffff <<  15 = 0xffff8000
(0050)          slli    x8, x8, 16          # (0050) x8 <- 0xffff8000 <<  16 = 0x80000000
(0054)          srai    x8, x8, 16          # (0054) x8 <- 0x80000000 >>> 16 = 0xffff8000
(0058)          srli    x8, x8, 15          # (0058) x8 <- 0xffff8000 >>  15 = 0x0001ffff
(005c)          slt     x3, x4, x6          # (005c) x3 <- 0xffffffff < 0x000002ff = 1
(0060)          jal     x0, s_x_s           # (0060) jump s_x_s
(1000) s_x_s:   li      a1, 0x7fffffff      # (1000) a               H = 0xfffffff8
(1004)          li      a1, 0x7fffffff      # (1004) a               H = 0xfffffff8
```

Waveform of Instruction TLB



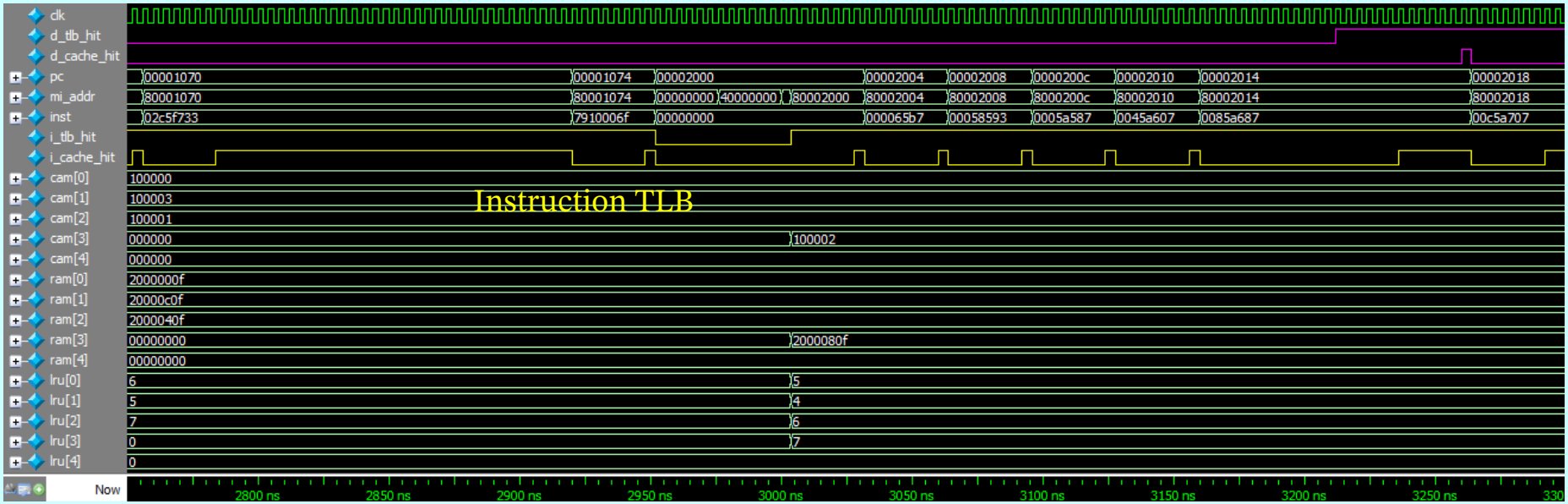

```
(1008)          li      a2, -15             # (1008) b                L = 0x8000000f
(100c)          mulh    a5,  a1, a2         # (100c) product high
(1010)          mul     a4,  a1, a2         # (1010) product low, fused with mulh
(1014) u_x_u:   li      a1, 0x7fffffff      # (1014) a                H = 0x7ffffffe
(1018) u_x_u:   li      a1, 0x7fffffff      # (1018) a                H = 0x7ffffffe
(101c)          li      a2, -1              # (101c) b                L = 0x80000001
(1020)          mulhu   a5,  a1, a2         # (1020) product high
(1024)          mul     a4,  a1, a2         # (1024) product low, fused with mulhu
(1028) s_x_u:   li      a1, 0x7fffffff      # (1028) a                H = 0x7ffffffe
(102c)          li      a1, 0x7fffffff      # (102c) a                H = 0x7ffffffe
(1030)          li      a2, -1              # (1030) b                L = 0x80000001
(1034)          mulhsu  a5,  a1, a2         # (1034) product high
```

Waveform of Instruction TLB



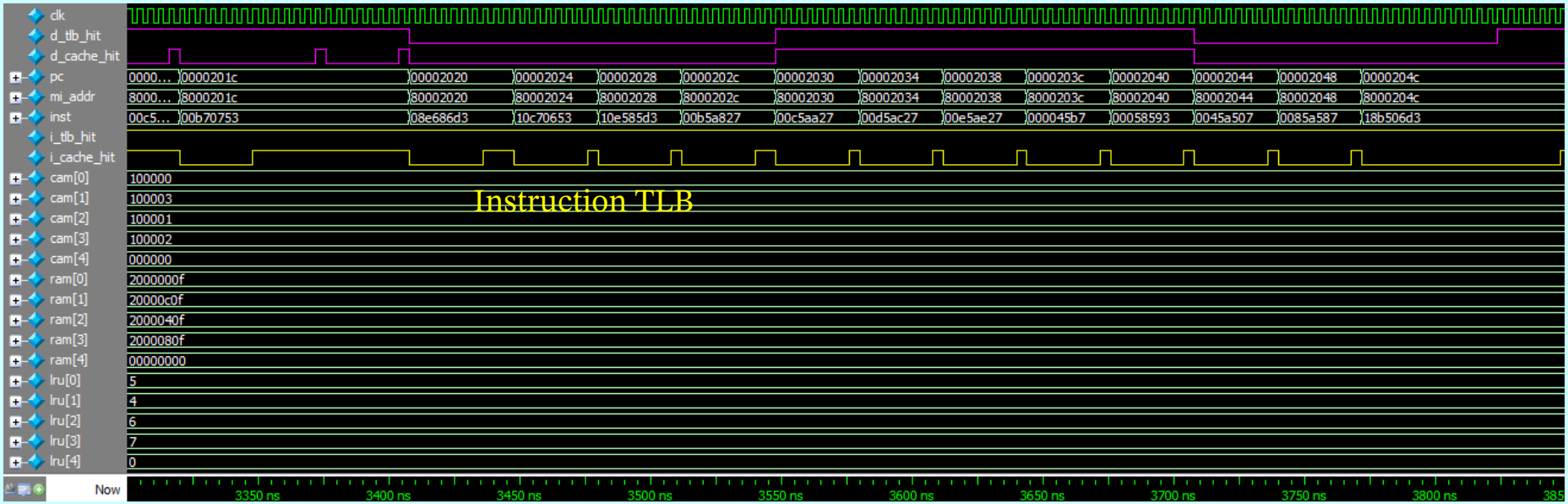

```
(1040)          li      a1, 0x7fffffff      # (1040) a                Q = 0x2aaaaaaa
(1044)          li      a2,   3             # (1044) b                R = 0x00000001
(1048)          div     a5,  a1, a2         # (1048) div signed
(104c)          rem     a4,  a1, a2         # (104c) rem signed, fused with div
(1050) sign1:   li      a1, 0xfffffff2      # (1050) a                Q = 0xfffffffc
(1054)          li      a2,   3             # (1054) b                R = 0xfffffffe
(1058)          div     a5,  a1, a2         # (1058) div signed
(105c)          rem     a4,  a1, a2         # (105c) rem signed, fused with div
(1060) unsign:  li      a1, 0xfffffff2      # (1060) a                Q = 0x55555550
(1064)          li      a2,   3             # (1064) b                R = 0x00000002
(1068)          divu    a5,  a1, a2         # (1068) div unsigned
(106c)          remu    a4,  a1, a2         # (106c) rem unsigned, fused with divu
```

Waveform of Instruction TLB



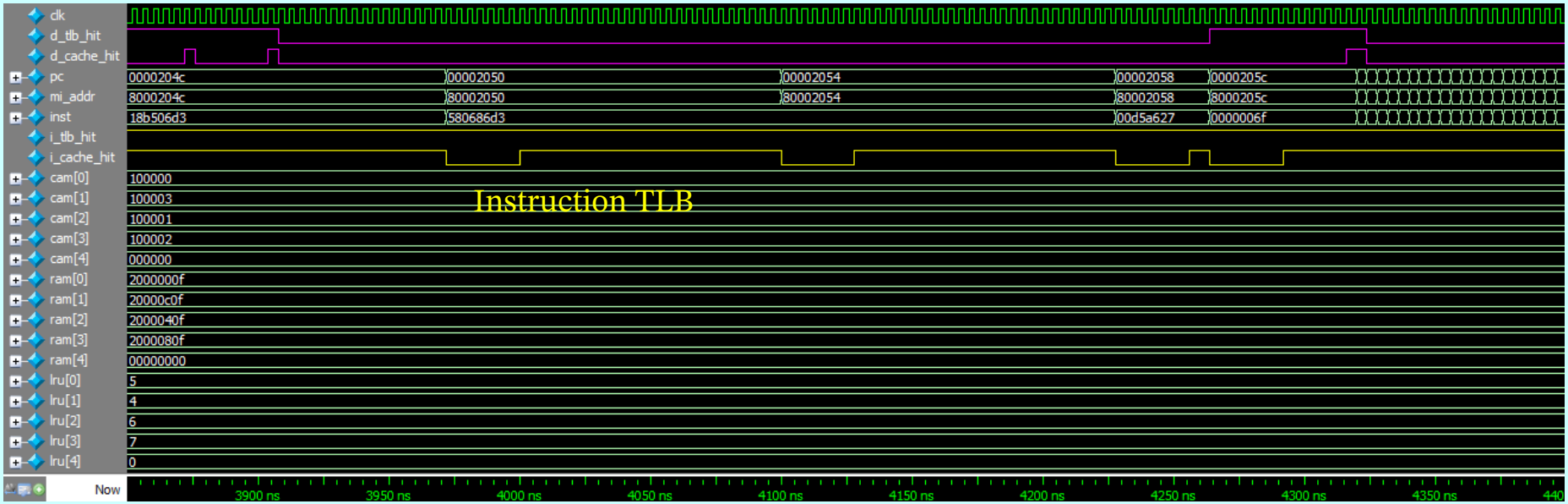

```
(1070)          jal     x0, fpu             # (1070) jump fpu
(1074)                                      # (1074) canceled
(2000) fpu:     lui     a1, %hi(array2)     # (2000) a1 <- array2: 0x00006000 virtual address
(2004)          addi    a1, a1, %lo(array2) # (2004) a1 <- array2: 0x00006000 virtual address
(2008)          flw     fa1,   0(a1)        # (2008) 0x40c00000
(200c)          flw     fa2,   4(a1)        # (200c) 0x41c00000
(2010)          flw     fa3,   8(a1)        # (2010) 0x43c00000
(2014)          flw     fa4,  12(a1)        # (2014) 0x47c00000
(2018)          fadd.s  fa4, fa4, fa1       # (2018) (47c00000) + (40c00000) = (47c00300)
(201c)          fsub.s  fa3, fa3, fa4       # (201c) (43c00000) - (47c00300) = (c7bf4300)
(2020)          fmul.s  fa2, fa4, fa2       # (2020) (47c00300) * (41c00000) = (4a100240)
(2024)          fmul.s  fa1, fa1, fa4       # (2024) (40c00000) * (47c00300) = (49100240)
```

Waveform of Instruction TLB



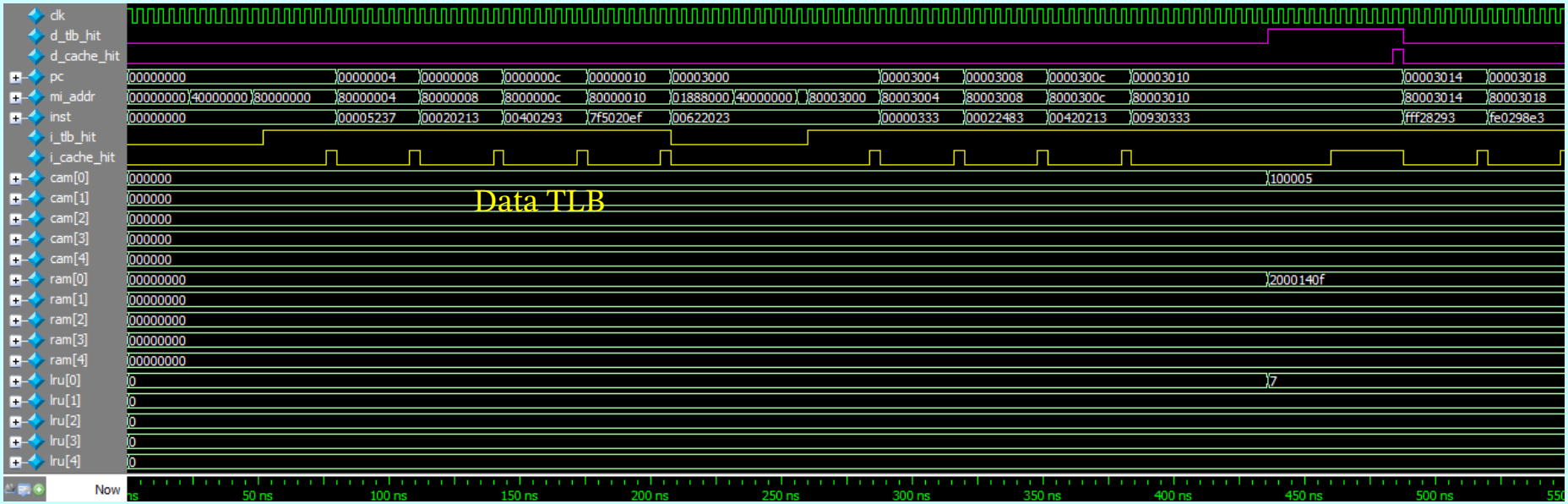

```
(201c)          fsub.s  fa3, fa3, fa4       # (201c) (43c00000) - (47c00300) = (c7bf4300)
(2020)          fmul.s  fa2, fa4, fa2       # (2020) (47c00300) * (41c00000) = (4a100240)
(2024)          fmul.s  fa1, fa1, fa4       # (2024) (40c00000) * (47c00300) = (49100240)
(2028)          fsw     fa1,  16(a1)        # (2028)
(202c)          fsw     fa2,  20(a1)        # (202c)
(2030)          fsw     fa3,  24(a1)        # (2030)
(2034)          fsw     fa4,  28(a1)        # (2034)
(2038)          lui     a1, %hi(array0)     # (2038) a1 <- array0: 0x00004000 virtual address
(203c)          addi    a1, a1, %lo(array0) # (203c) a1 <- array0: 0x00004000 virtual address
(2040)          flw     fa0,   4(a1)        # (2040)  40800000
(2044)          flw     fa1,   8(a1)        # (2044)  40000000
(2048)          fdiv.s  fa3, fa0, fa1       # (2048) (40800000) / (40000000) = (40000000)
```

Waveform of Instruction TLB



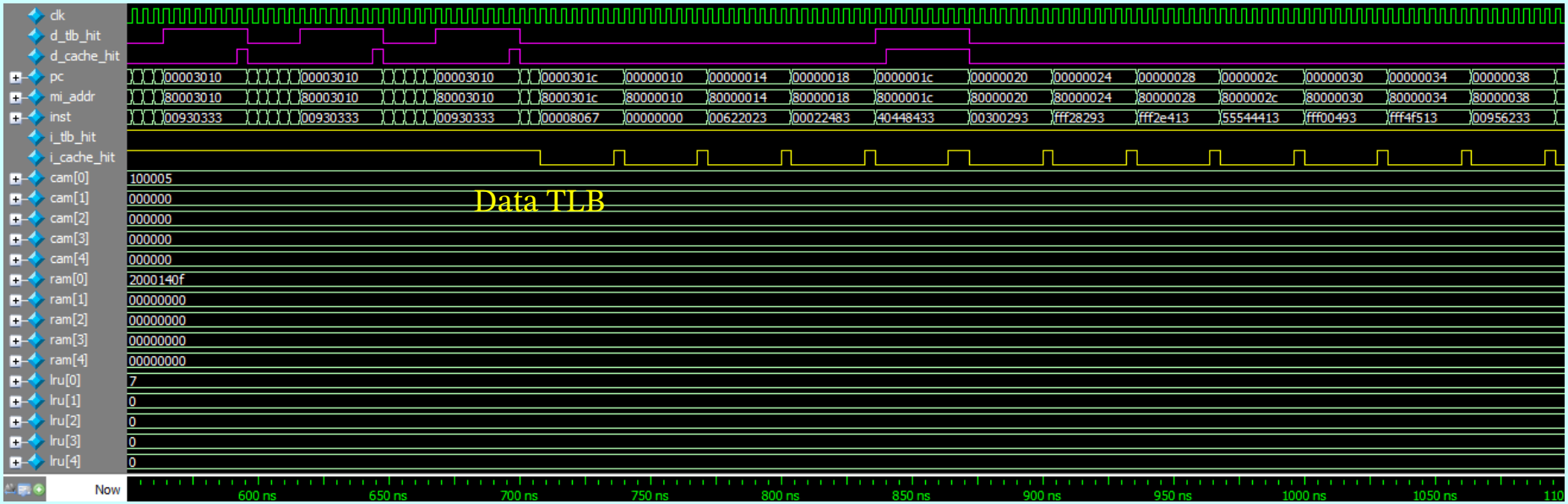

```
(204c)          fsqrt.s fa3, fa3          # (204c) (40000000)      sqrt    = (3fb504f3)
(2050)          fsqrt.s fa3, fa3          # (2050) (3fb504f3)      sqrt    = (3f9837f0)
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
```

Waveform of Data TLB



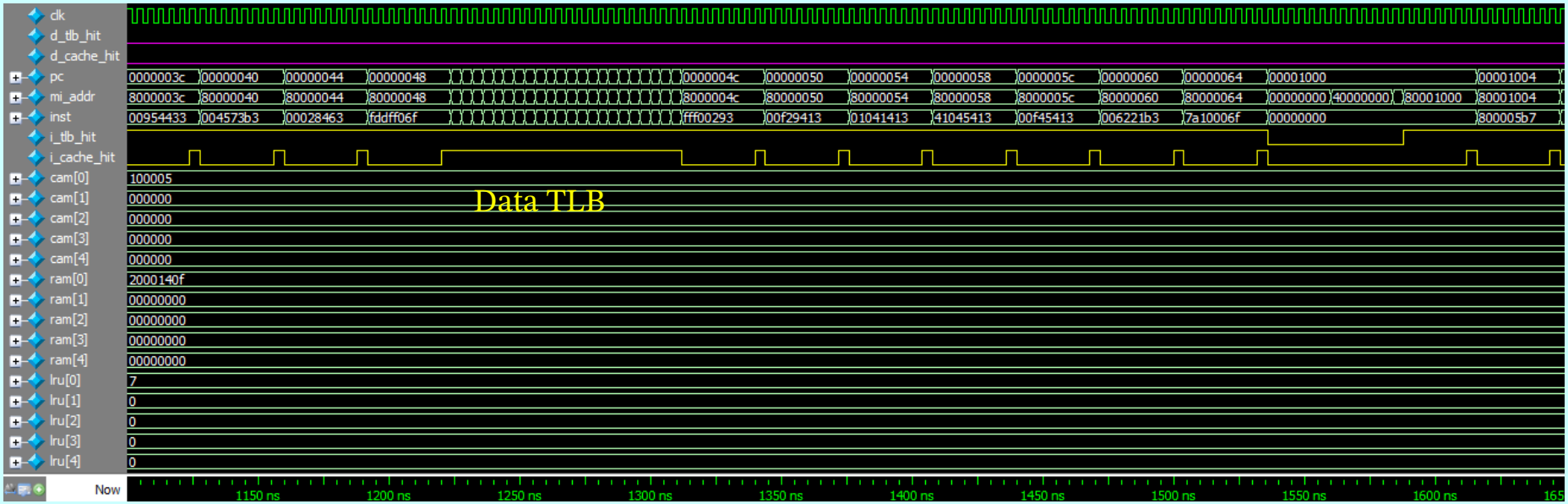

```
(0000) main:   lui     x4, %hi(array1)     # (0000) x4 <- array1: 0x5000 virtual address
(0004)         addi    x4, x4, %lo(array1) # (0004) x4 <- array1: 0x5000 virtual address
(0008)         addi    x5, x0,  4          # (0008) x5 <- 4
(000c) call:   jal     x1, sum             # (000c) x1 <- 0x10 (return address), call sum
(0010)         sw      x6, 0(x4)           # (0010) memory[x4+0] <- x6, canceled
(3000) sum:    add     x6, x0, x0          # (3000) x6 <- 0 (subroutine entry)
(3004) loop:   lw      x9, 0(x4)           # (3004) x9 <- memory[x4+0]
(3008)         addi    x4, x4,  4          # (3008) x4 <- x4 + 4 (address+4)
(300c)         add     x6, x6, x9          # (300c) x6 <- x6 + x9 (sum)
(3010)         addi    x5, x5, -1          # (3010) x5 <- x5 - 1 (counter--)
(3014)         bne     x5, x0, loop        # (3014) if x5 != 0, goto loop
(3018)         ret     x1                  # (3018) return from subroutine 100 1101 00
```

Waveform of Data TLB



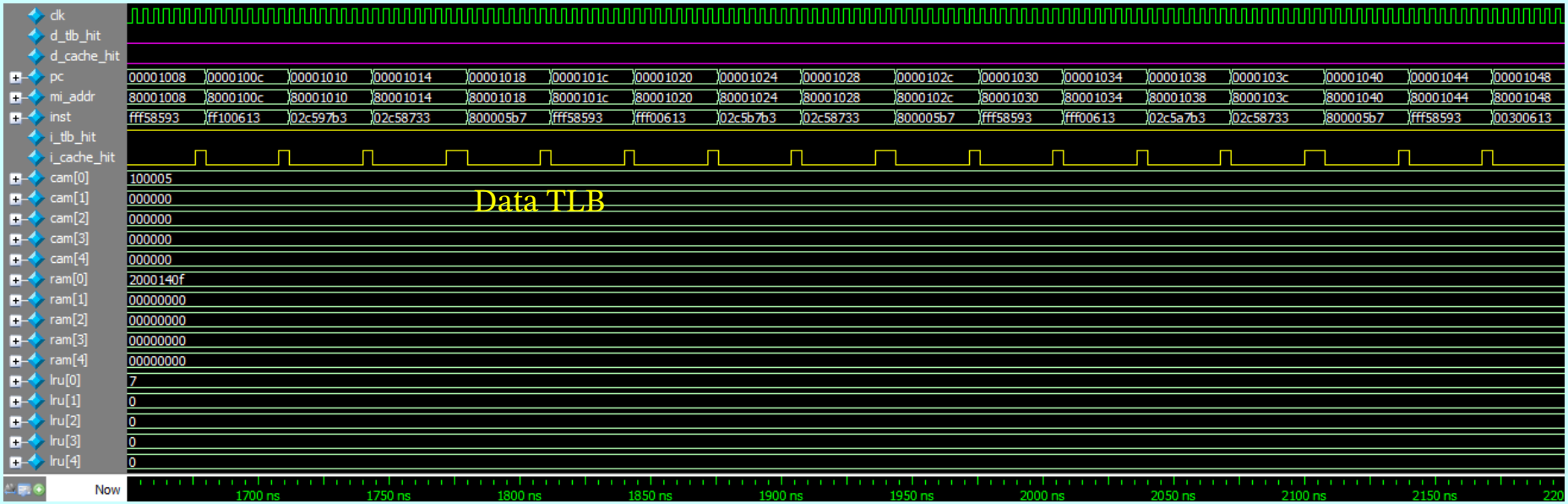

```
(3018)          ret     x1              # (3018) return from subroutine 100 1101 00
(301c)                                  # (301c) canceled
(0010)          sw      x6, 0(x4)       # (0010) memory[x4+0] <- x6
(0014)          lw      x9, 0(x4)       # (0014) x6 <- memory[x4+0]
(0018)          sub     x8, x9, x4      # (0018) x8 <- x9 - x4
(001c)          addi    x5, x0,  3      # (001c) x5 <- 3
(0020) loop2:   addi    x5, x5, -1      # (0020) x5 <- x5 - 1
(0024)          ori     x8, x5, 0xfff   # (0024) x8 <- x5 | 0xffffffff = 0xffffffff
(0028)          xori    x8, x8, 0x555   # (0028) x8 <- x8 ^ 0x00000555 = 0xfffffaaa
(002c)          addi    x9, x0, -1      # (002c) x9 <- 0xffffffff
(0030)          andi    x10,x9, 0xfff   # (0030) x10<- x9 & 0xffffffff = 0xffffffff
(0034)          or      x4, x10, x9     # (0034) x4 <- x10 | x9 = 0xffffffff
```

Waveform of Data TLB



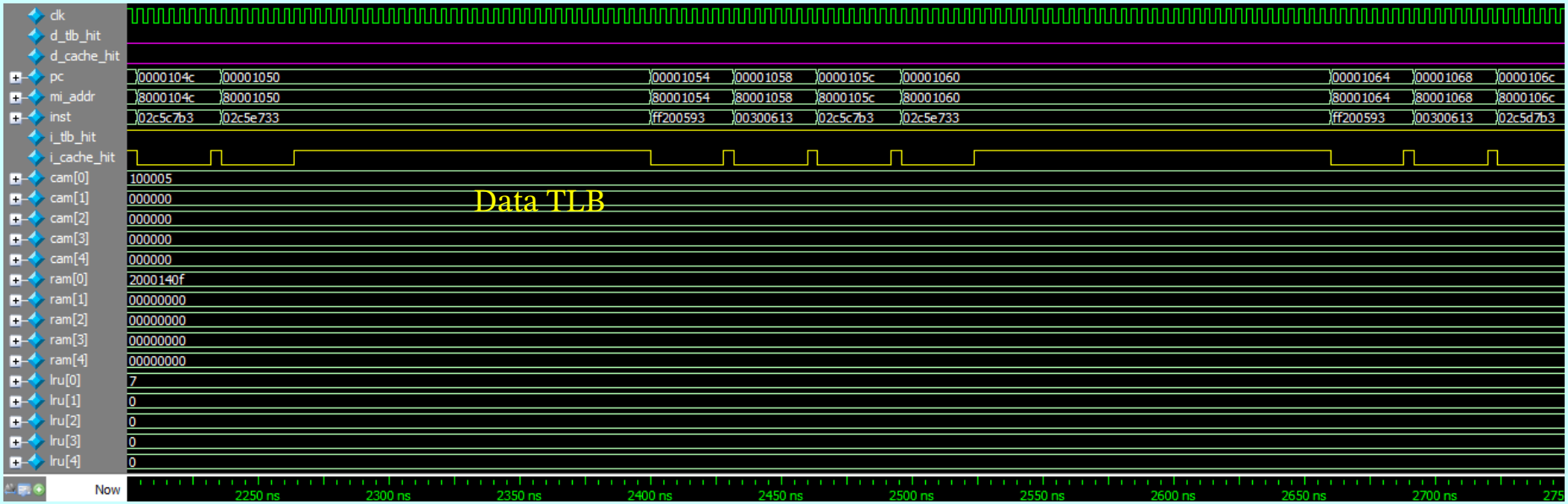

```
(003c)          and     x7, x10, x4         # (003c) x7 <- x10 & x4 = 0xffffffff
(0040)          beq     x5, x0, shift       # (0040) if x5 = 0, goto shift
(0044)          jal     x0, loop2           # (0044) jump loop2
(0048) shift:   addi    x5, x0, -1          # (0048) x5 <- 0xffffffff
(004c)          slli    x8, x5, 15          # (004c) x8 <- 0xffffffff <<  15 = 0xffff8000
(0050)          slli    x8, x8, 16          # (0050) x8 <- 0xffff8000 <<  16 = 0x80000000
(0054)          srai    x8, x8, 16          # (0054) x8 <- 0x80000000 >>> 16 = 0xffff8000
(0058)          srli    x8, x8, 15          # (0058) x8 <- 0xffff8000 >>  15 = 0x0001ffff
(005c)          slt     x3, x4, x6          # (005c) x3 <- 0xffffffff < 0x000002ff = 1
(0060)          jal     x0, s_x_s           # (0060) jump s_x_s
(1000) s_x_s:   li      a1, 0x7fffffff      # (1000) a               H = 0xfffffff8
(1004)          li      a1, 0x7fffffff      # (1004) a               H = 0xfffffff8
```

Waveform of Data TLB



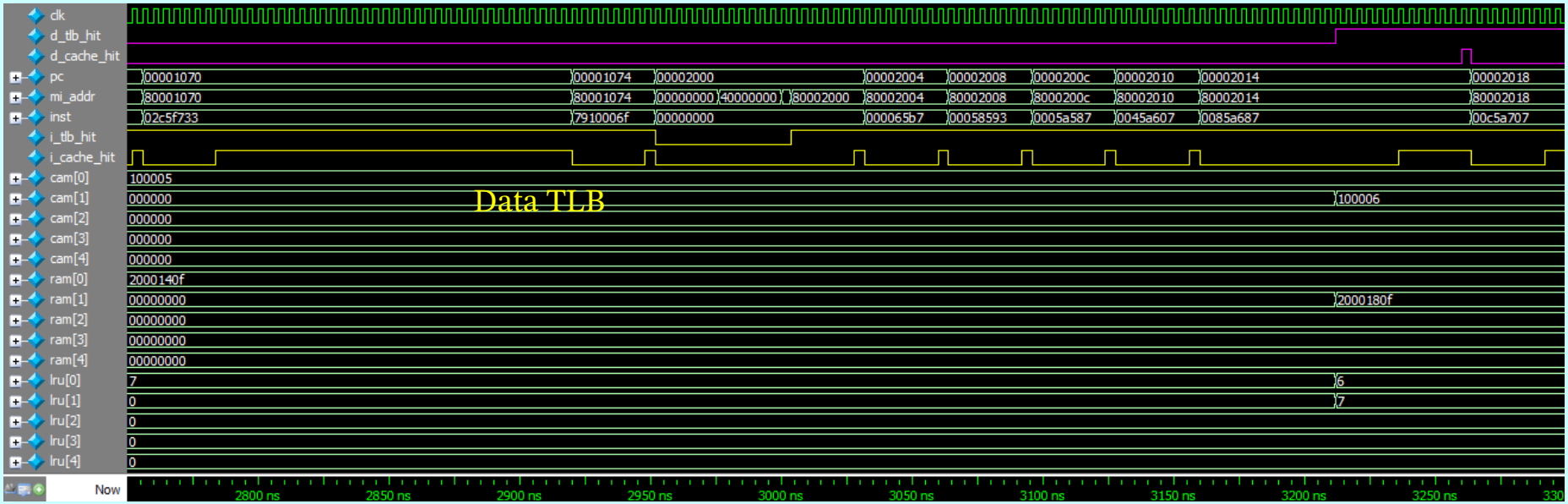

```
(1008)          li      a2, -15             # (1008) b              L = 0x8000000f
(100c)          mulh    a5,  a1, a2         # (100c) product high
(1010)          mul     a4,  a1, a2         # (1010) product low, fused with mulh
(1014) u_x_u:   li      a1, 0x7fffffff      # (1014) a              H = 0x7ffffffe
(1018) u_x_u:   li      a1, 0x7fffffff      # (1018) a              H = 0x7ffffffe
(101c)          li      a2, -1              # (101c) b              L = 0x80000001
(1020)          mulhu   a5,  a1, a2         # (1020) product high
(1024)          mul     a4,  a1, a2         # (1024) product low, fused with mulhu
(1028) s_x_u:   li      a1, 0x7fffffff      # (1028) a              H = 0x7ffffffe
(102c)          li      a1, 0x7fffffff      # (102c) a              H = 0x7ffffffe
(1030)          li      a2, -1              # (1030) b              L = 0x80000001
(1034)          mulhsu  a5,  a1, a2         # (1034) product high
```

Waveform of Data TLB



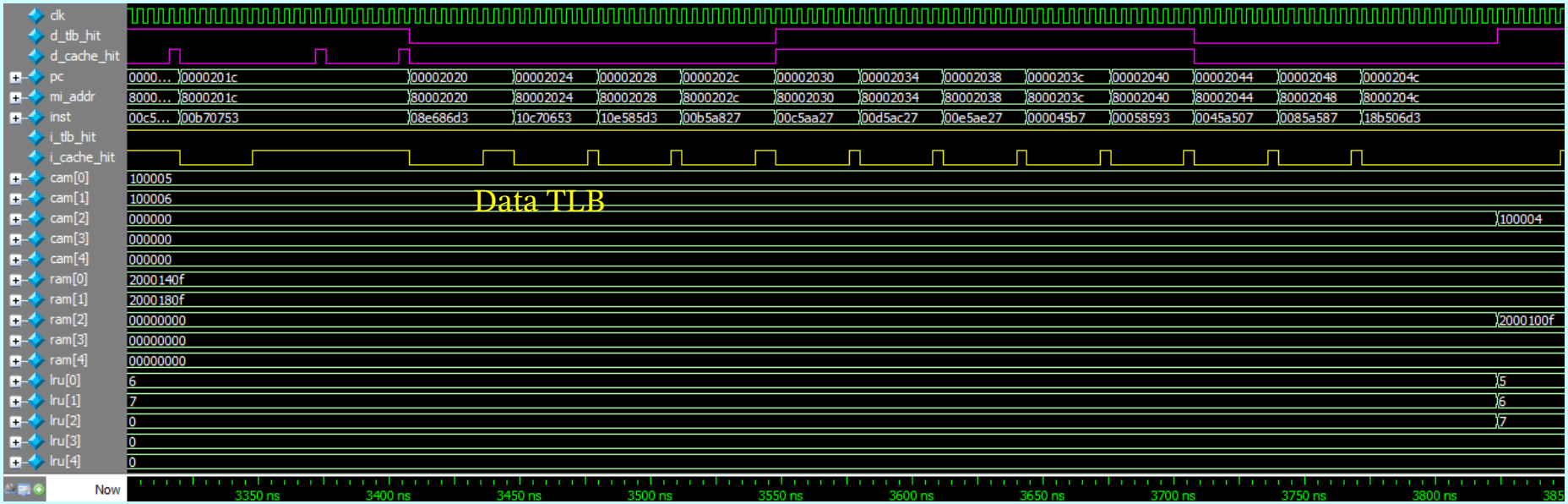

```
(1040)          li      a1, 0x7fffffff      # (1040) a                Q = 0x2aaaaaaa
(1044)          li      a2,   3             # (1044) b                R = 0x00000001
(1048)          div     a5,  a1, a2         # (1048) div signed
(104c)          rem     a4,  a1, a2         # (104c) rem signed, fused with div
(1050) sign1:   li      a1, 0xfffffff2      # (1050) a                Q = 0xfffffffc
(1054)          li      a2,   3             # (1054) b                R = 0xfffffffe
(1058)          div     a5,  a1, a2         # (1058) div signed
(105c)          rem     a4,  a1, a2         # (105c) rem signed, fused with div
(1060) unsign:  li      a1, 0xfffffff2      # (1060) a                Q = 0x55555550
(1064)          li      a2,   3             # (1064) b                R = 0x00000002
(1068)          divu    a5,  a1, a2         # (1068) div unsigned
(106c)          remu    a4,  a1, a2         # (106c) rem unsigned, fused with divu
```

Waveform of Data TLB



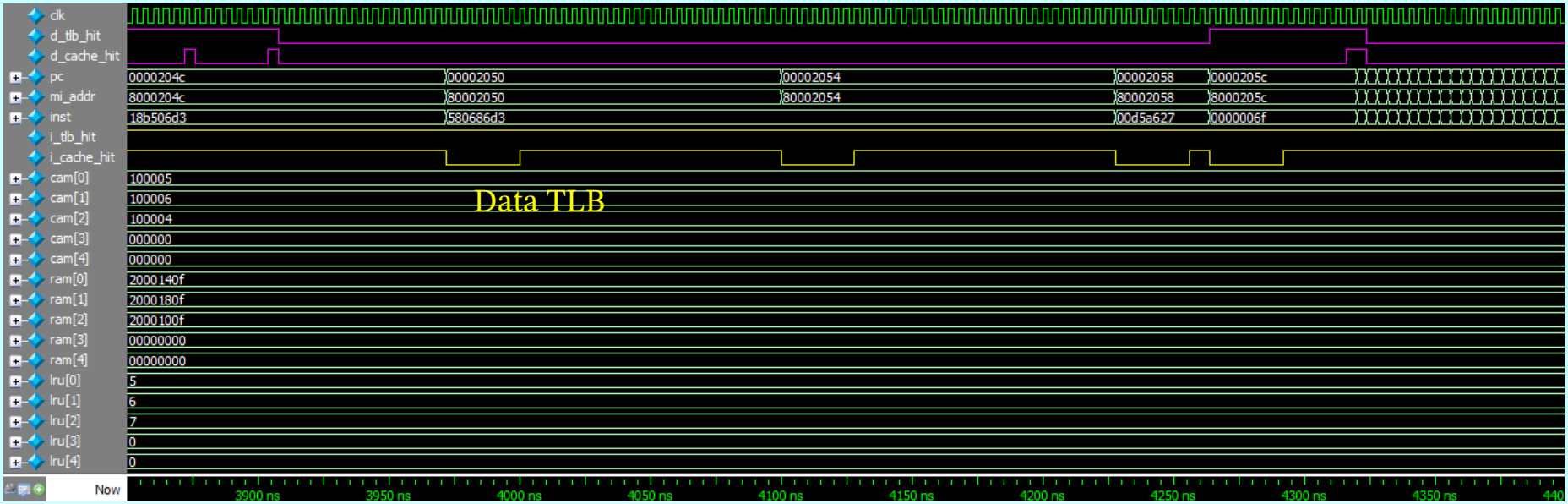

```
(1070)          jal     x0, fpu             # (1070) jump fpu
(1074)                                      # (1074) canceled
(2000) fpu:     lui     a1, %hi(array2)     # (2000) a1 <- array2: 0x00006000 virtual address
(2004)          addi    a1, a1, %lo(array2) # (2004) a1 <- array2: 0x00006000 virtual address
(2008)          flw     fa1,   0(a1)        # (2008) 0x40c00000
(200c)          flw     fa2,   4(a1)        # (200c) 0x41c00000
(2010)          flw     fa3,   8(a1)        # (2010) 0x43c00000
(2014)          flw     fa4,  12(a1)        # (2014) 0x47c00000
(2018)          fadd.s  fa4, fa4, fa1       # (2018) (47c00000) + (40c00000) = (47c00300)
(201c)          fsub.s  fa3, fa3, fa4       # (201c) (43c00000) - (47c00300) = (c7bf4300)
(2020)          fmul.s  fa2, fa4, fa2       # (2020) (47c00300) * (41c00000) = (4a100240)
(2024)          fmul.s  fa1, fa1, fa4       # (2024) (40c00000) * (47c00300) = (49100240)
```

Waveform of Data TLB




```
(201c)          fsub.s  fa3, fa3, fa4       # (201c) (43c00000) - (47c00300) = (c7bf4300)
(2020)          fmul.s  fa2, fa4, fa2       # (2020) (47c00300) * (41c00000) = (4a100240)
(2024)          fmul.s  fa1, fa1, fa4       # (2024) (40c00000) * (47c00300) = (49100240)
(2028)          fsw     fa1,  16(a1)        # (2028)
(202c)          fsw     fa2,  20(a1)        # (202c)
(2030)          fsw     fa3,  24(a1)        # (2030)
(2034)          fsw     fa4,  28(a1)        # (2034)
(2038)          lui     a1, %hi(array0)     # (2038) a1 <- array0: 0x00004000 virtual address
(203c)          addi    a1, a1, %lo(array0) # (203c) a1 <- array0: 0x00004000 virtual address
(2040)          flw     fa0,   4(a1)        # (2040)  40800000
(2044)          flw     fa1,   8(a1)        # (2044)  40000000
(2048)          fdiv.s  fa3, fa0, fa1       # (2048) (40800000) / (40000000) = (40000000)
```

Waveform of Data TLB




```
(204c)          fsqrt.s fa3, fa3          # (204c) (40000000)      sqrt    = (3fb504f3)
(2050)          fsqrt.s fa3, fa3          # (2050) (3fb504f3)      sqrt    = (3f9837f0)
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
(2054) finish: jal      x0, finish        # (2054) dead loop
(2058)                                    # (2058) canceled
```

課題 XI (100 点 + 100 点)

1. Explain the cache, MMU, and TLB, and why the cache, MMU, and TLB are required in modern computers.
2. Suppose h = 98%, t_c = 1 ns, and t_m = 6 ns, calculate the effective memory access time and speedup.
3. Referring to P33, calculate the total RAM bits required by a direct-mapped cache: CPU address: 32 bits; block index: 11 bits; block size: 16 bytes. And what is the cache size (in KB)?
4. Option (+100 点): Design a pipelined CPU with instruction cache & TLB and data cache & TLB.