Pipelined CPU with Caches and TLBs

Seungwan Noh

Pusan National University

Department of Electronics Engineering

Contents

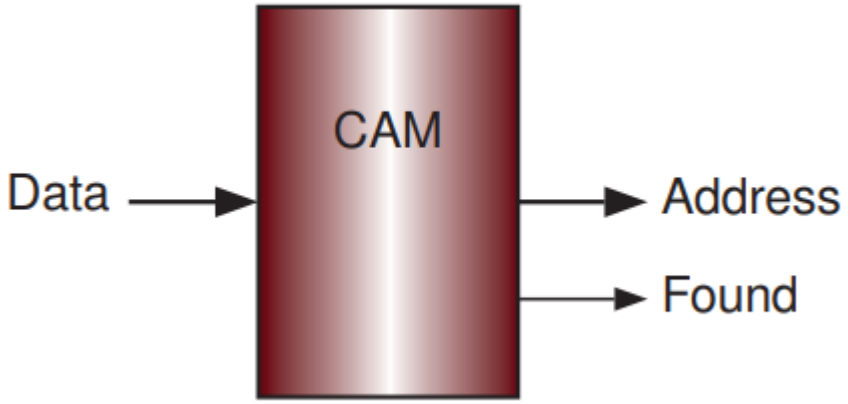
- Memory Hierarchy
- Virtual Memory and TLB
- TLB-Based MIPS Memory Management
- Design of Pipelined CPU with Caches and TLB in Verilog HDL
- Simulation
- References

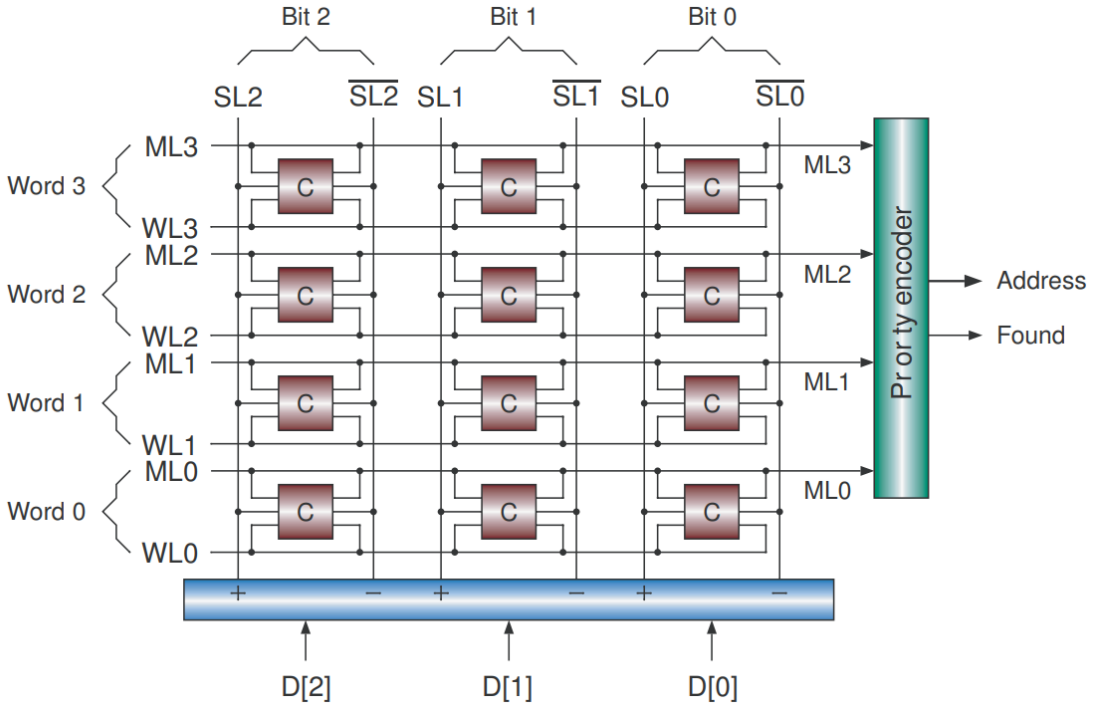
Memory

- Memory is a temporary place for storing programs.
- SRAM
  - ✓Fast and expensive (caches and TLBs).
- DRAM
  - ✓Large and inexpensive (main memory).
- ROM
  - ✓Read only memory (initial program or firmware).
- CAM
  - ✓Content addressable memory.
  - ✓Mainly used to design fully associative cache or TLB.

Content Addressable Memory

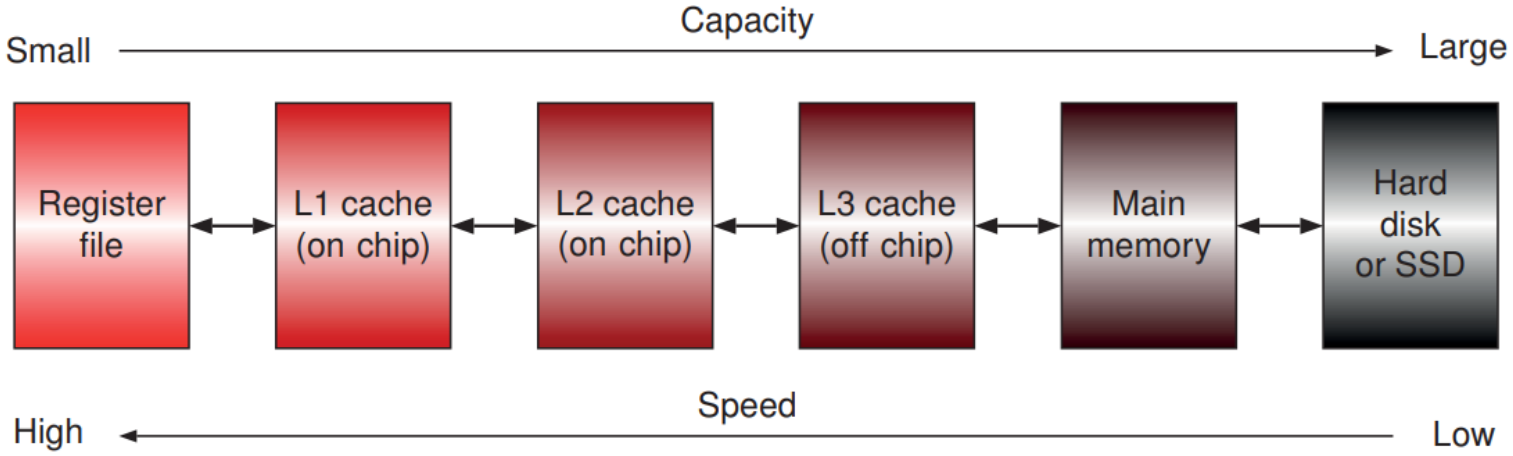
- CAM is a very special memory.
- CAM searches the entire memory to see given data word is stored anywhere in it.
- If the data word is found, CAM returns the address.





Memory Hierarchy

- The register file can be considered as the fastest memory.
- Caches store partial data of memory.
- L1 and L2 caches are on-chip, L3 is an off-chip.
- Main memory stores programs that are being executed.
- Hard disk or SSD has the largest capacity but is lowest speed.
- It store files with main memory or provide users virtual memory.

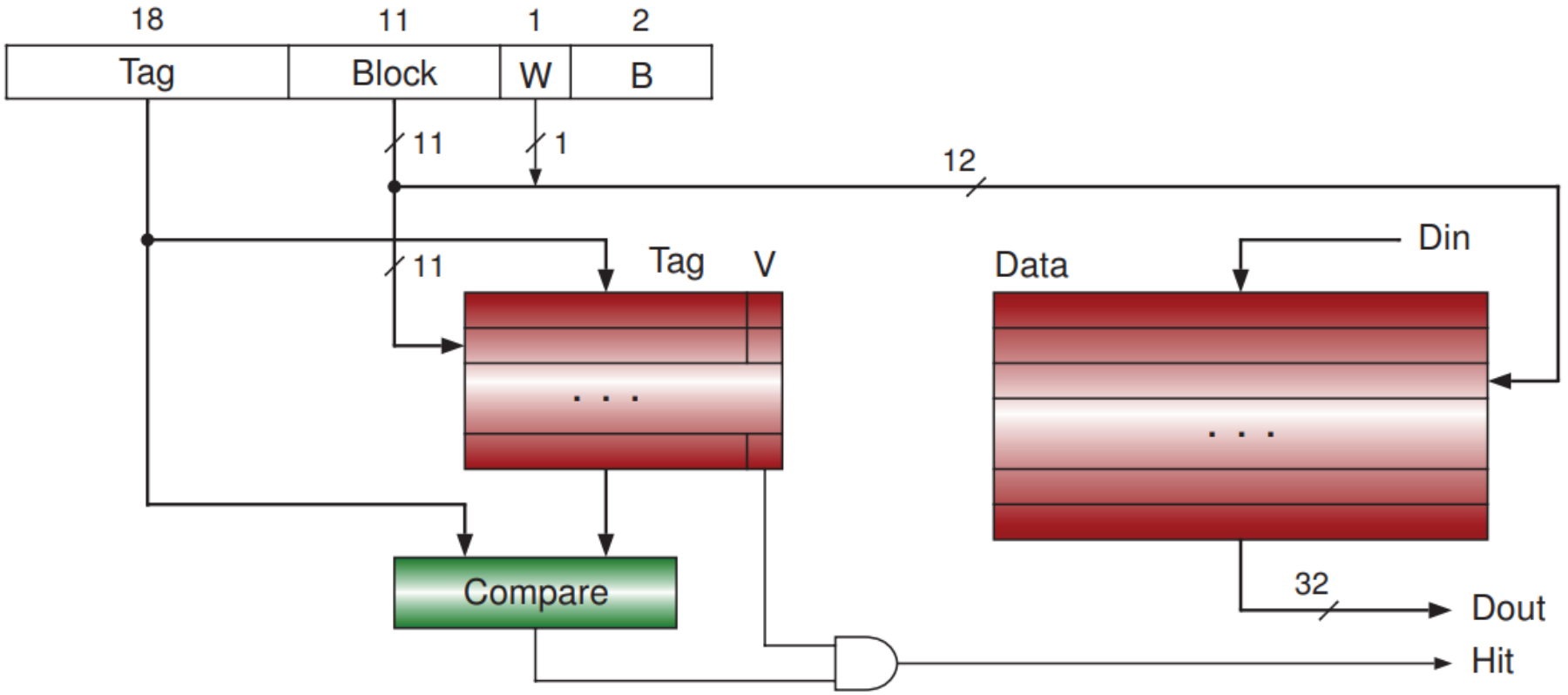


Cache

- Caches in the computer field is a fast storage for storing data that are likely to be used again.
- The benefit of using a cache comes from the program's temporal locality and spatial locality.

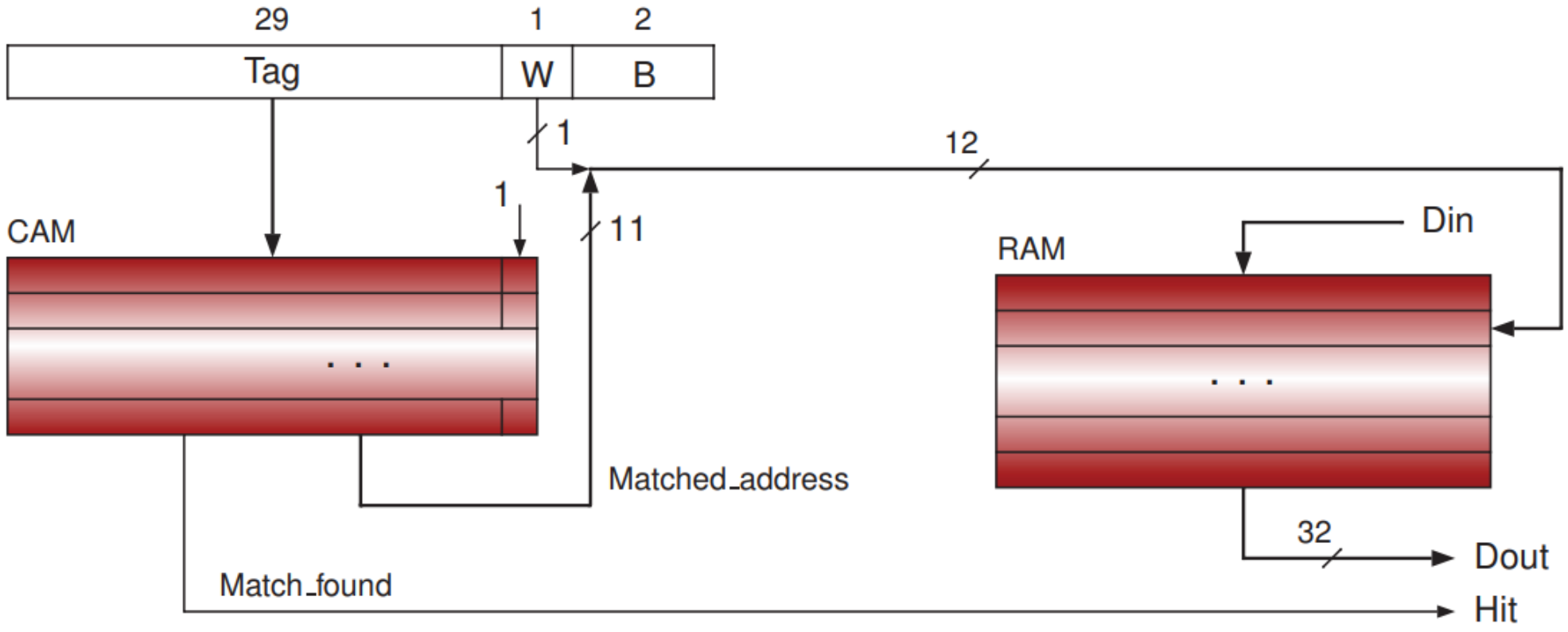
Cache

- Direct mapping cache



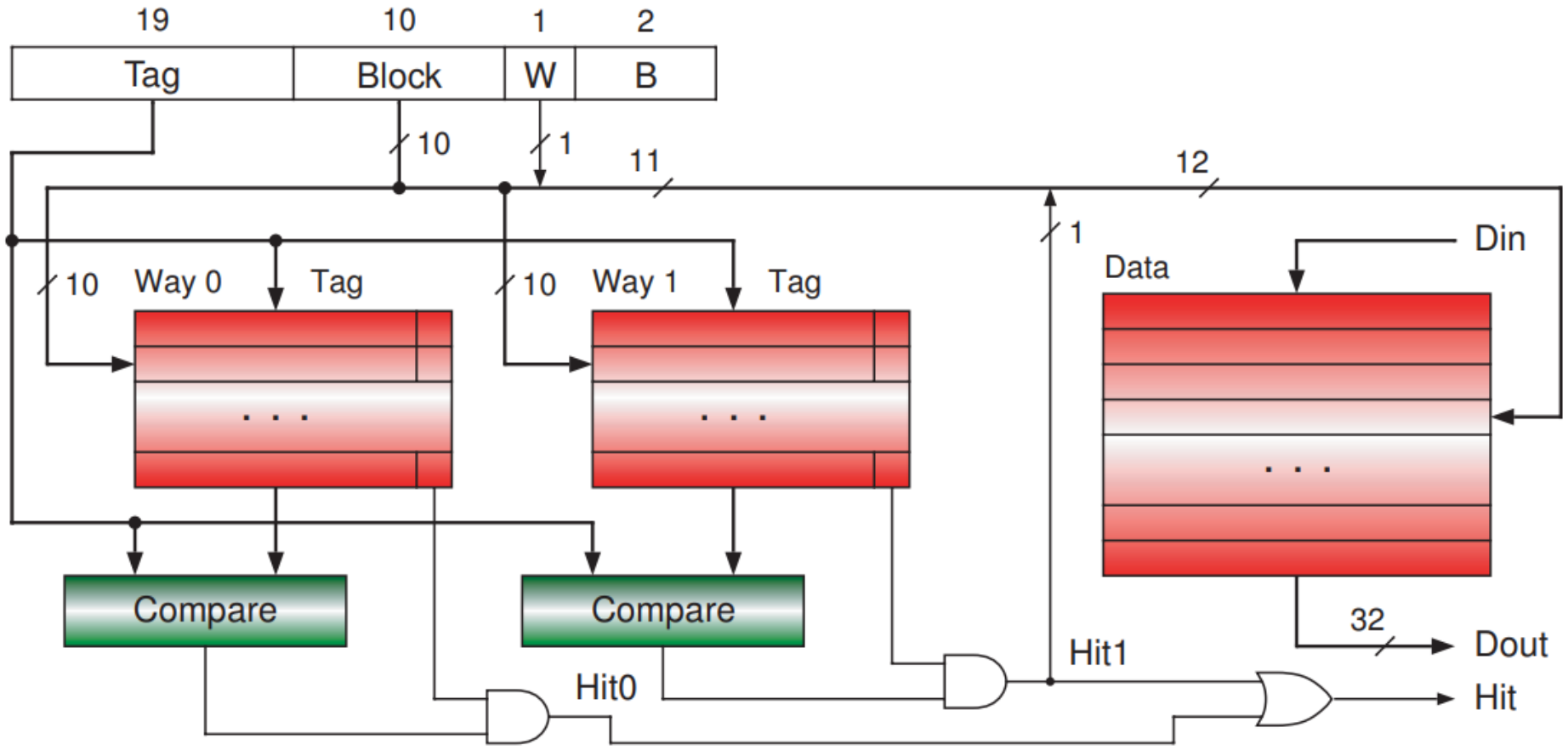
Cache

- Fully associative mapping



Cache

- Set associative mapping



Cache

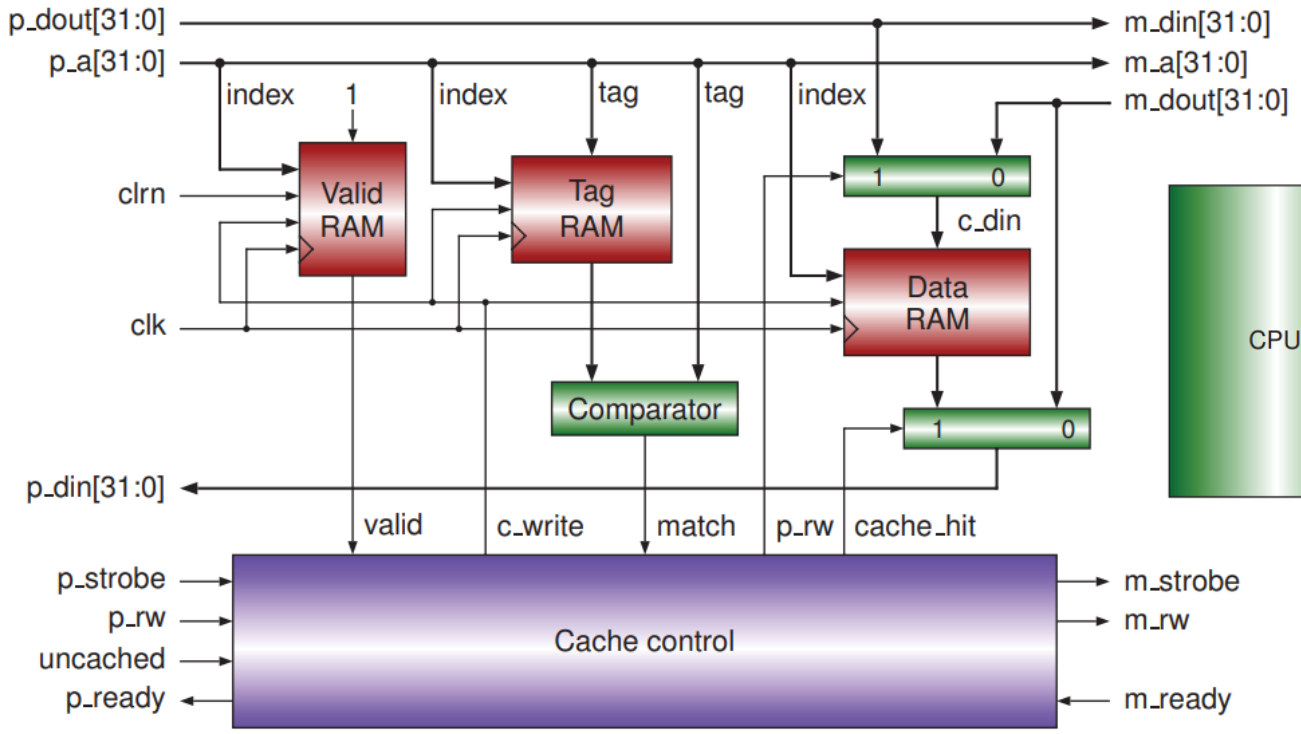
- Cache block replacement algorithms
- LRU replacement algorithm
- Random replacement algorithm
- FIFO replacement algorithm

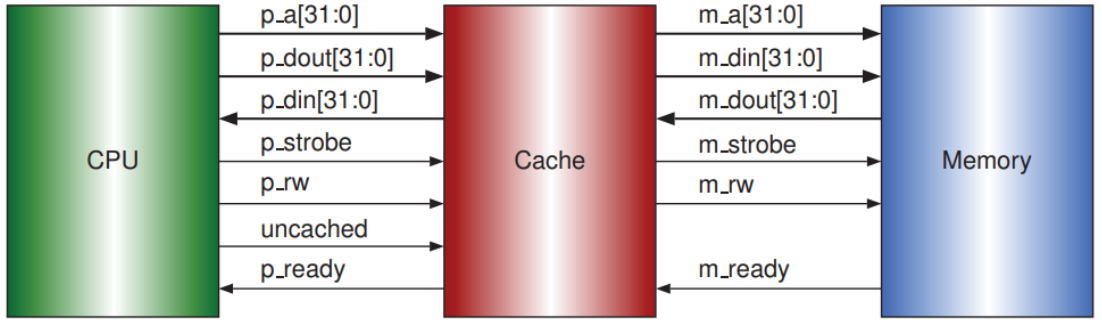
Cache

- Cache write policies
- Cache hit
  - ✓write through; update both the cache and the memory
  - ✓write back; update the cache only, when it relaced update the memory
- Cache miss
  - ✓write allocate
  - ✓no write allocate

Data Cache Design in Verilog HDL

- Direct mapping, write through, and write allocate





Data Cache Design in Verilog HDL

- Virtual memory is not a real memory.
- It allows processes to use more memory than the real memory.
- When process access a virtual memory, MMU translates the virtual address into a physical address.
  - ✓To speed up this translation, TLB is fabricated in CPU.

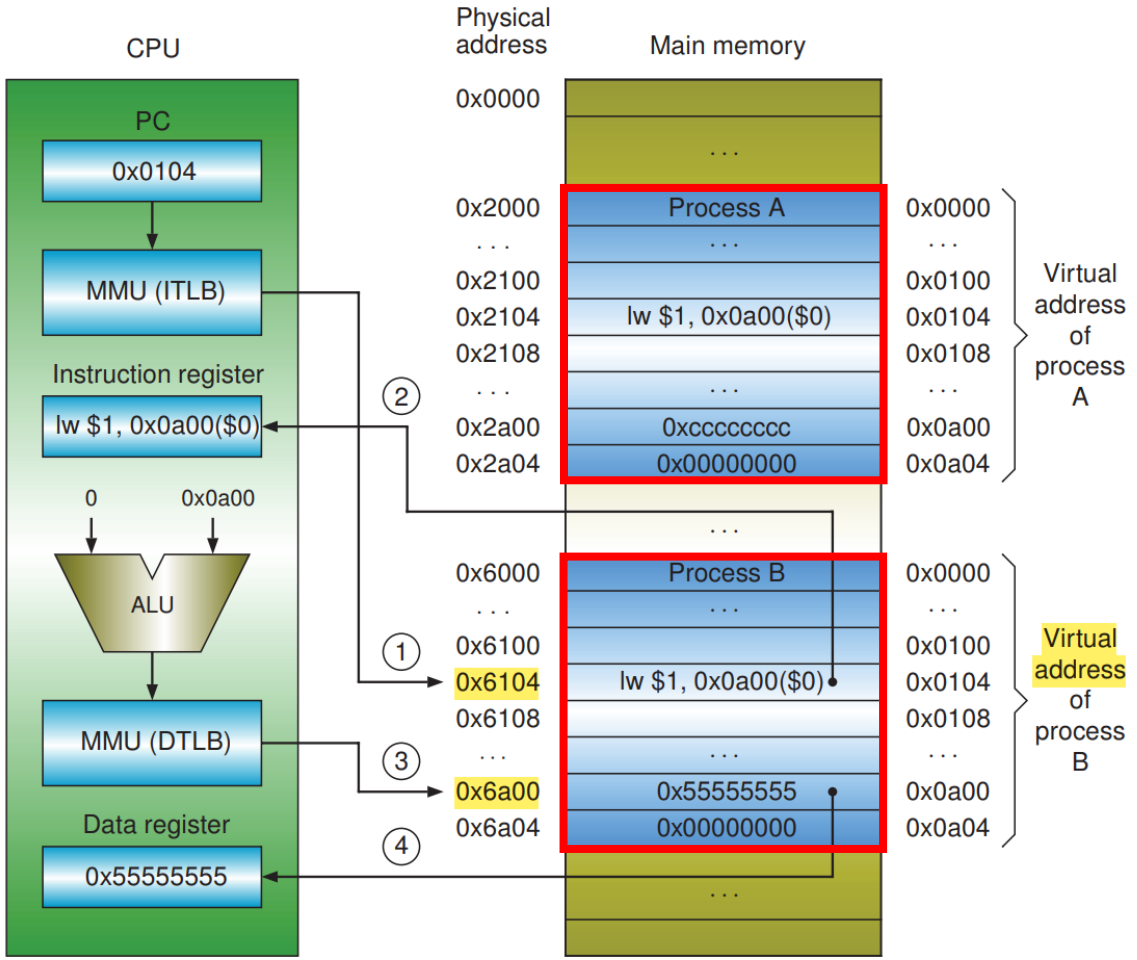
Virtual Memory and TLB

- The main memory is a real storage in which programs stored.
- Main memory is also called physical memory or real memory.
- Multiple processes can be resided in the main memory simultaneously.
- The process use a virtual address to access its virtual memory.

Process A		Process B	
Virtual address	Instruction or data	Virtual address	Instruction or data
0x0000:	...	0x0000:	...
...	...	...	...
0x0104:	lw $1, 0x0a00($0)	0x0104:	lw $1, 0x0a00($0)
...	...	...	...
0x0a00:	0xcccccccc	0x0a00:	0x55555555
0x0a04:	0x00000000	0x0a04:	0x00000000

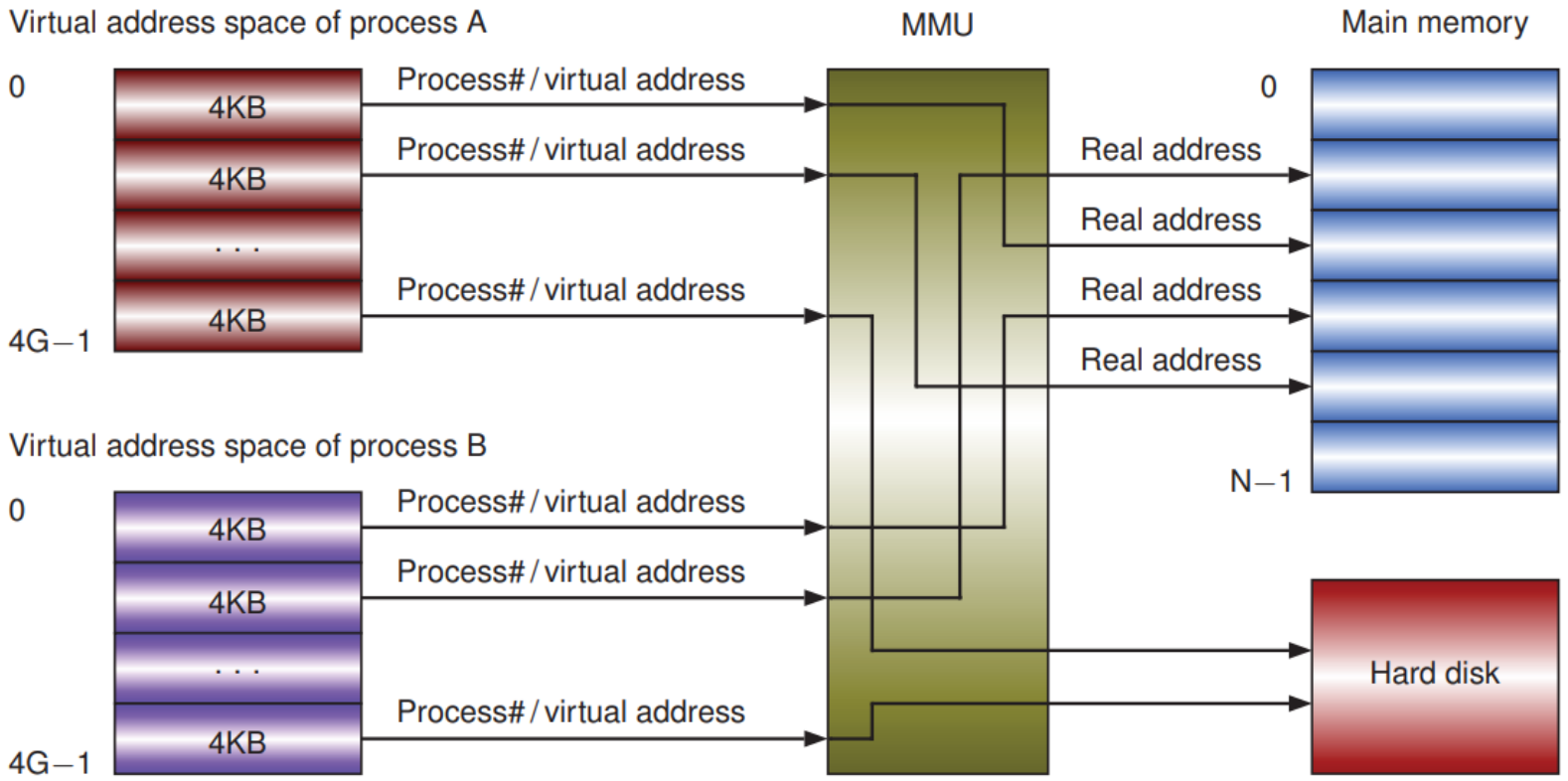
Virtual Memory and TLB

- Two processes reside in main memory



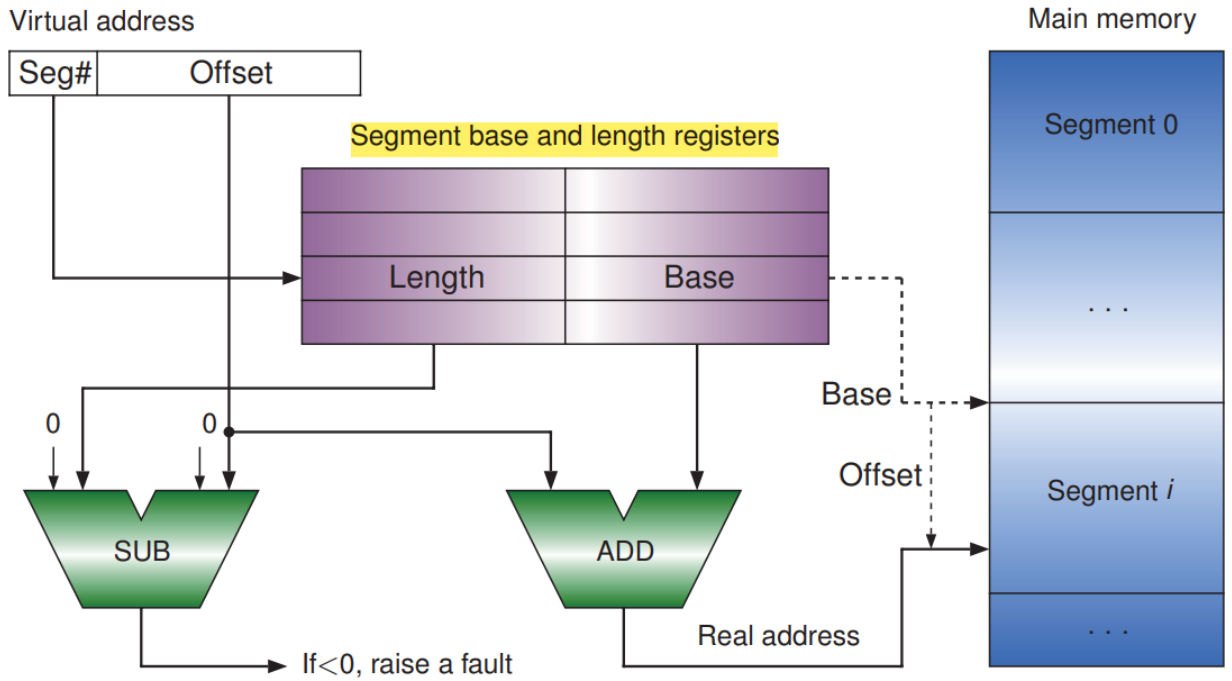
Virtual Memory and TLB

- Concept of virtual memory management



Virtual Memory and TLB

- **Segmentation management**
- With segmentation management, the main memory is divided into segments based on the natural divisions of a program.
- The size of a memory segment is generally not fixed.

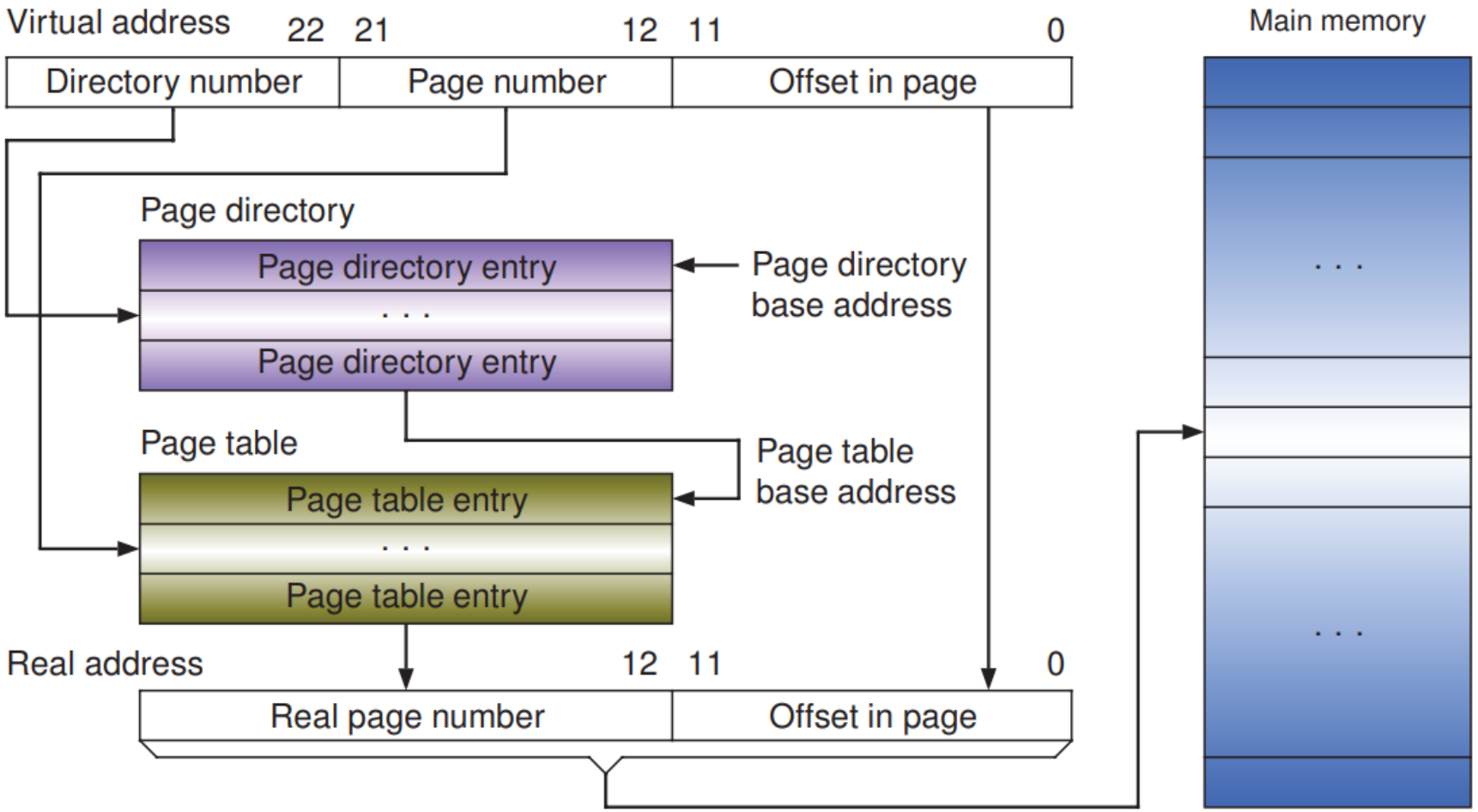


Virtual Memory and TLB

- **Paging management**
- With paging management, both the virtual memory and real memory are divided into pages of fixed size.
- Page is the smallest unit with which the MMU maps a virtual page to a physical page.
  - ✓Translate VPN to RPN
- Translation is done with a page table.
- To prevent page table takes a large memory, we use two-level page tables.
- The first-level page table is called page directory; it stores the physical base address of the (second-level) page table.

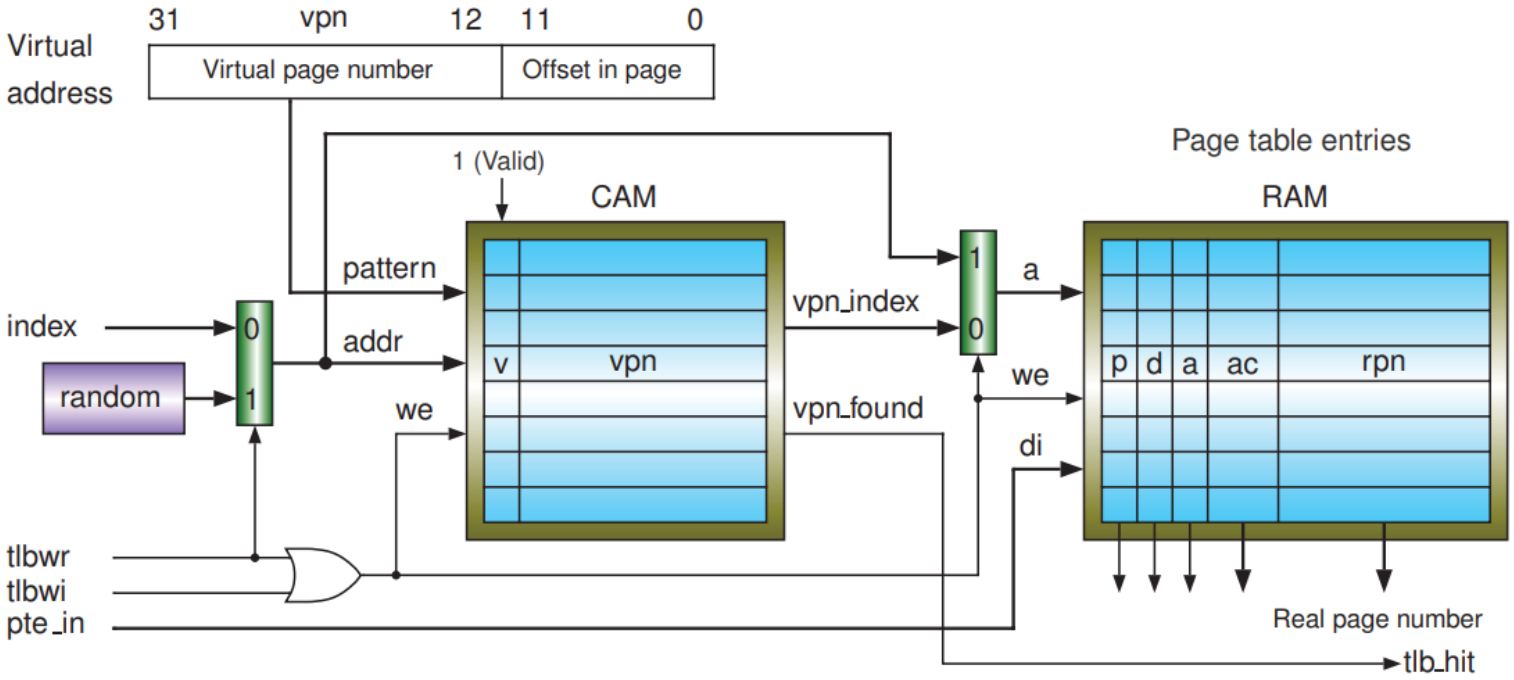
Virtual Memory and TLB

- Two-level paging management

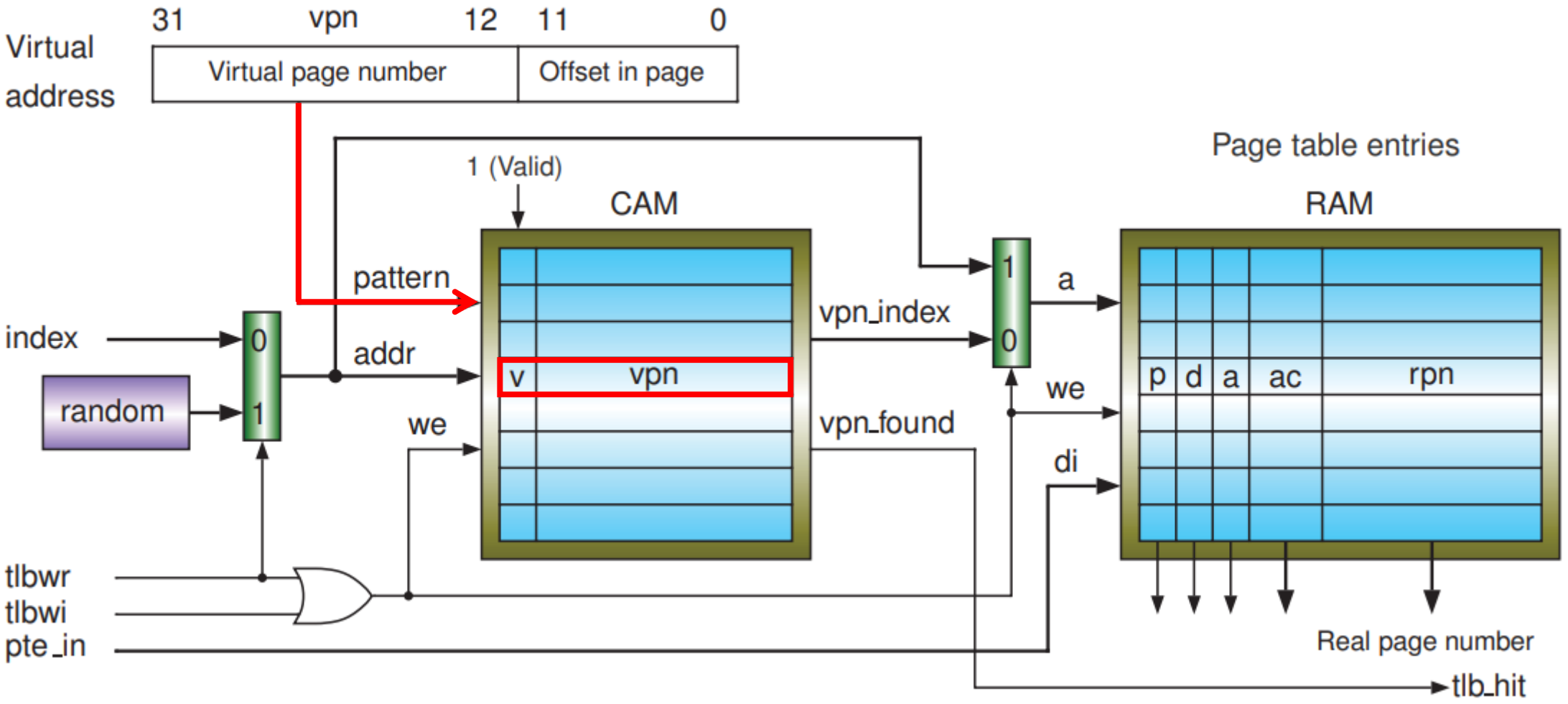


Virtual Memory and TLB

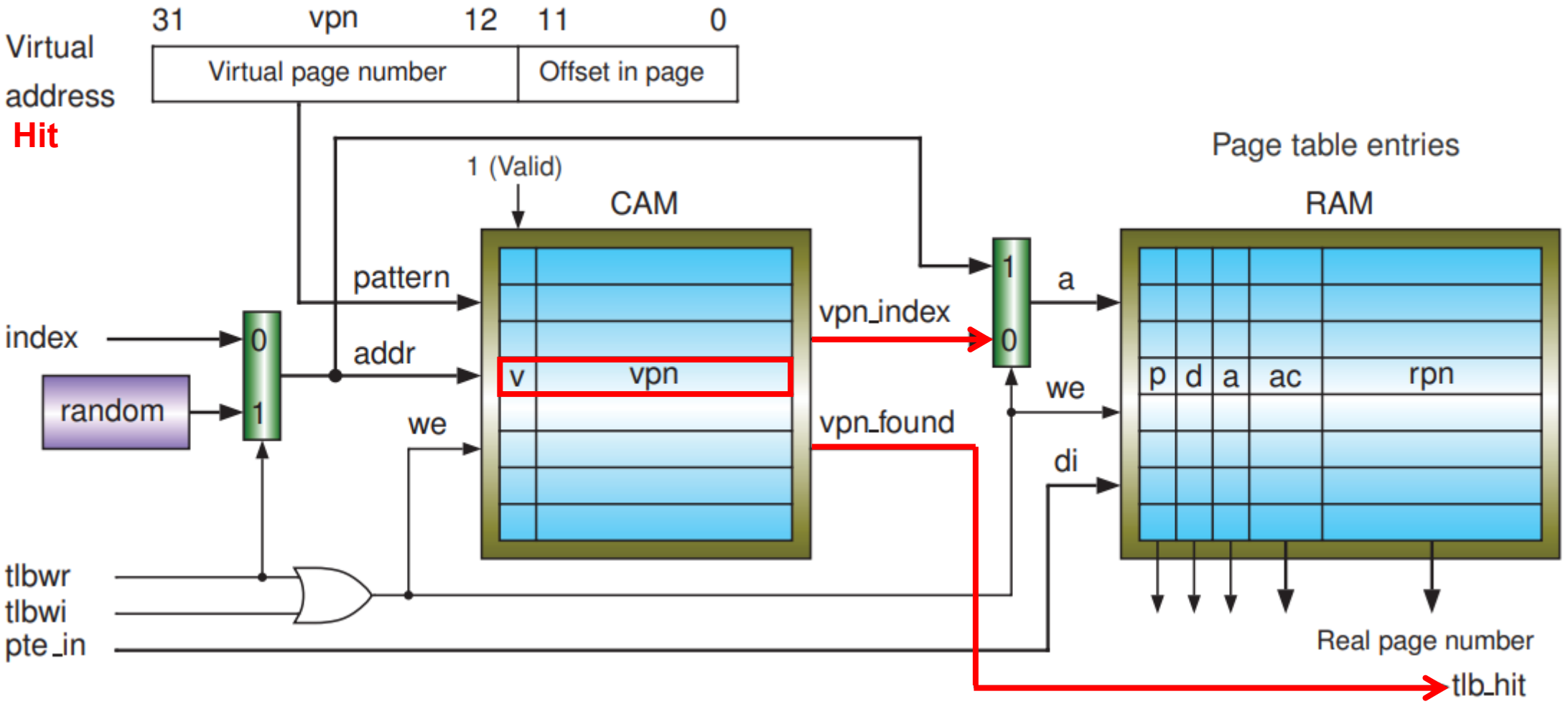
- To speed up the translation, CPUs fabricate **TLBs**.
- The data cache stores data blocks in cache RAM, while the TLB stores RPNs.
- CAM is used to search for the VPN of the virtual address.



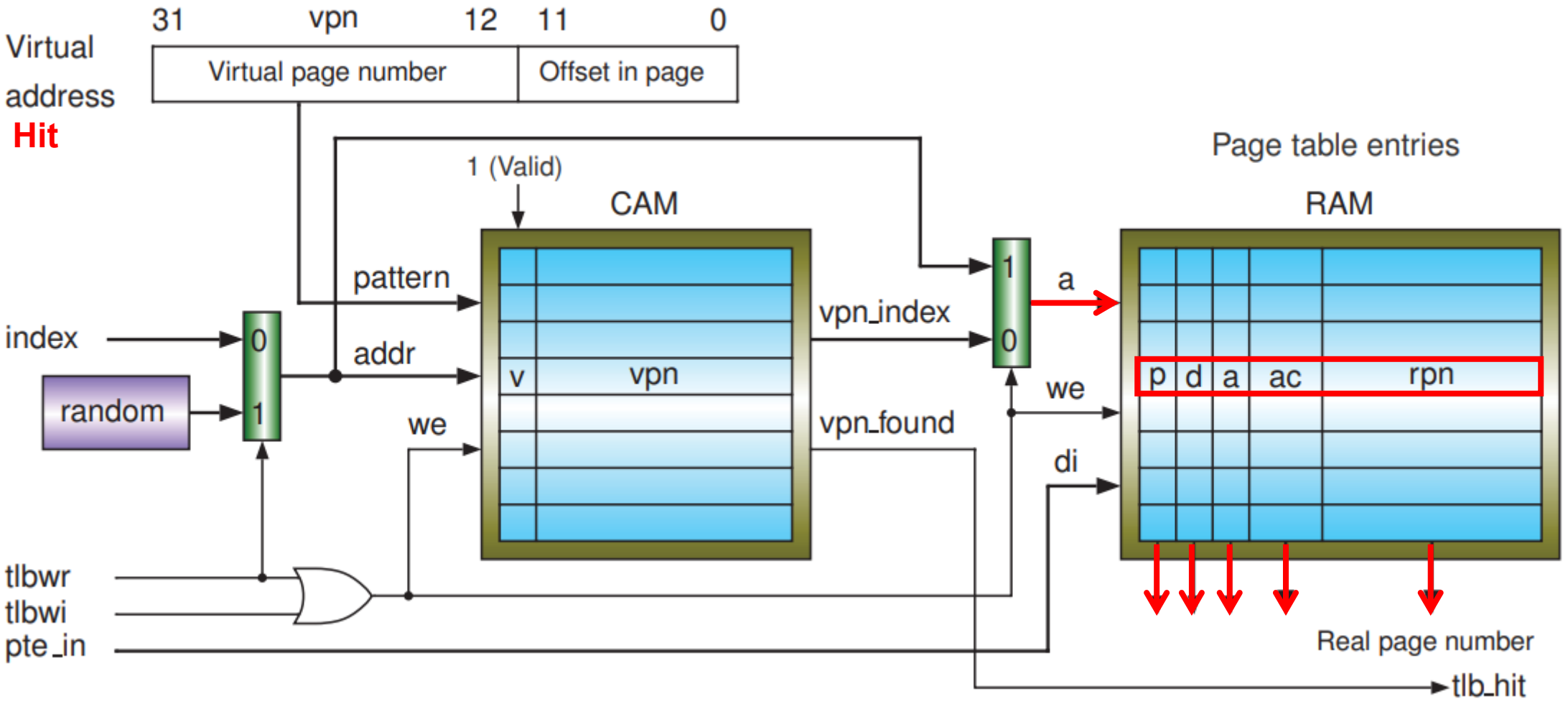
Virtual Memory and TLB



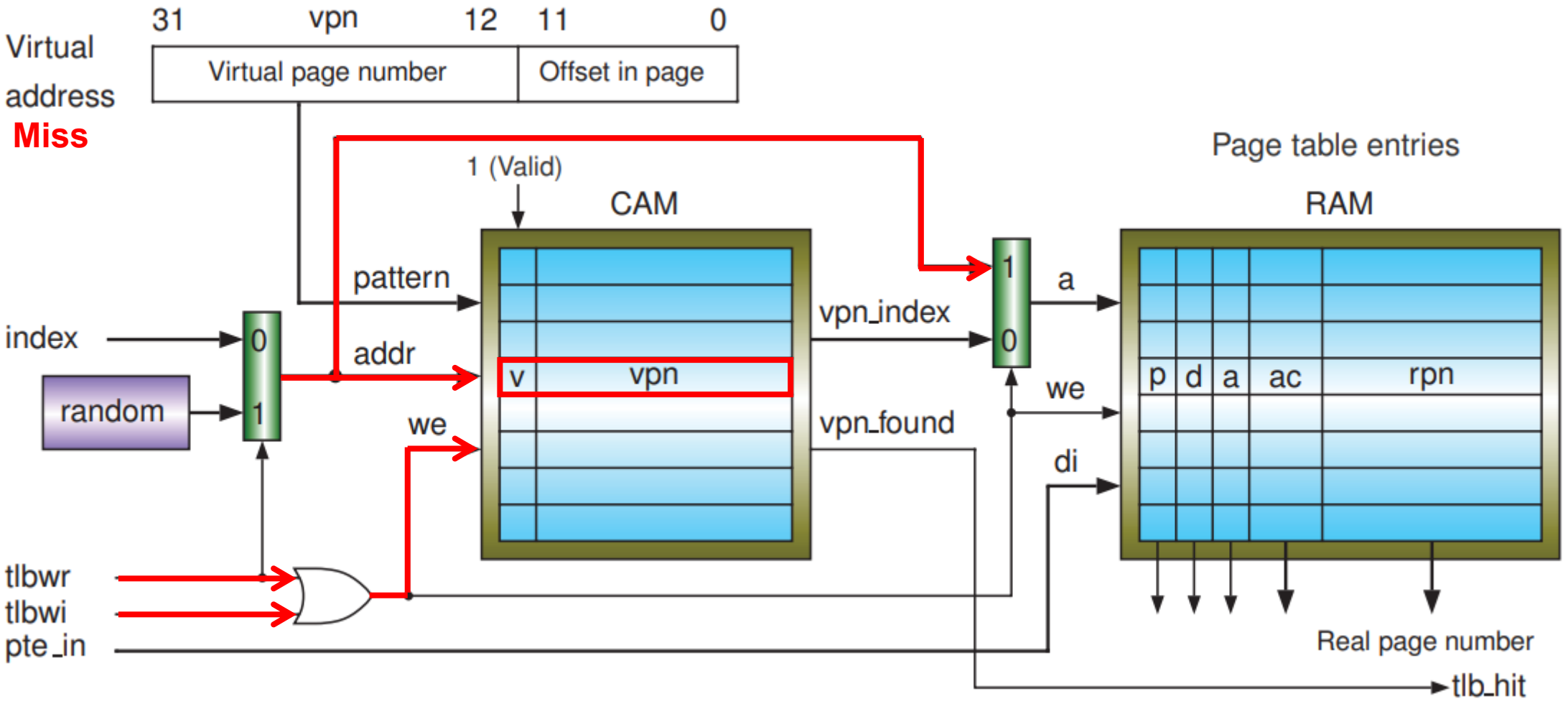
Virtual Memory and TLB



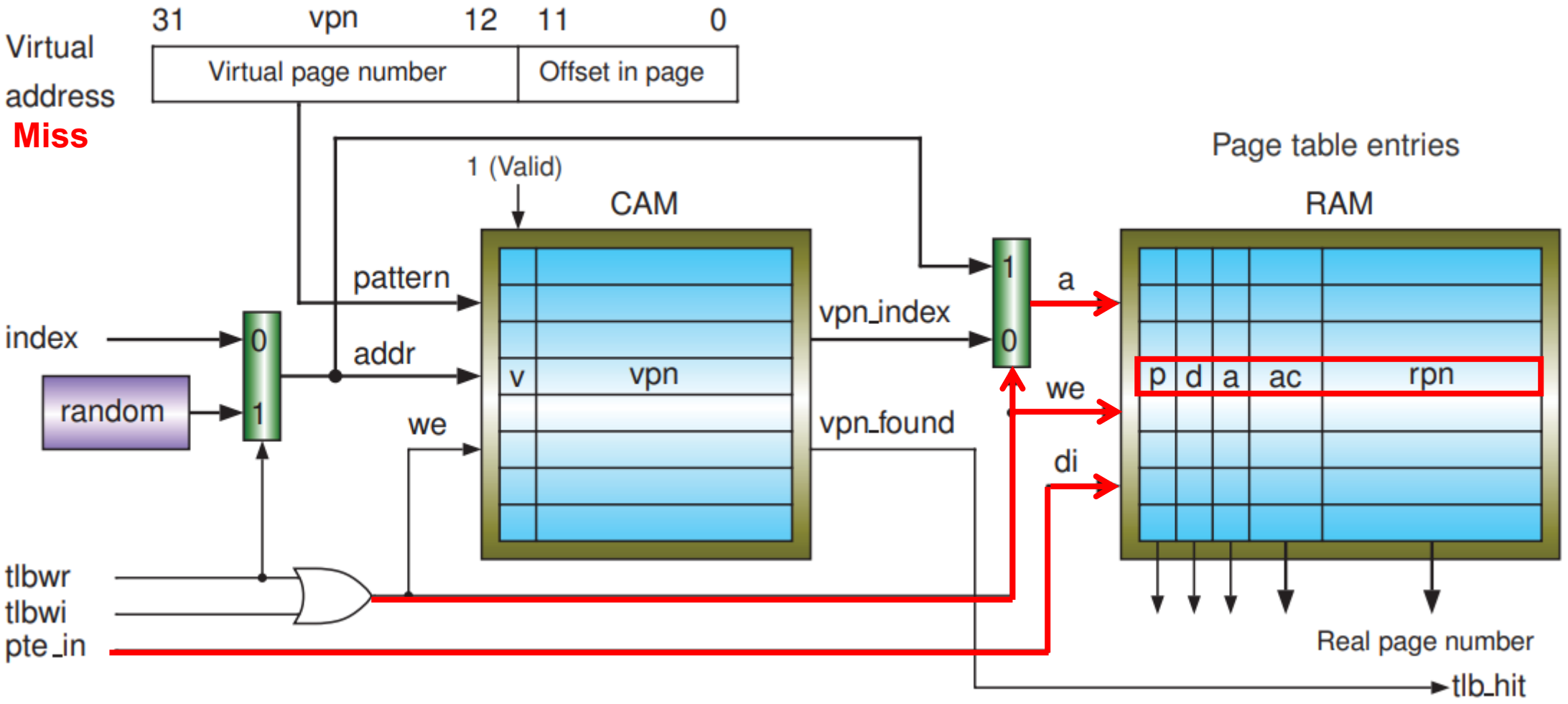
Virtual Memory and TLB



Virtual Memory and TLB



Virtual Memory and TLB

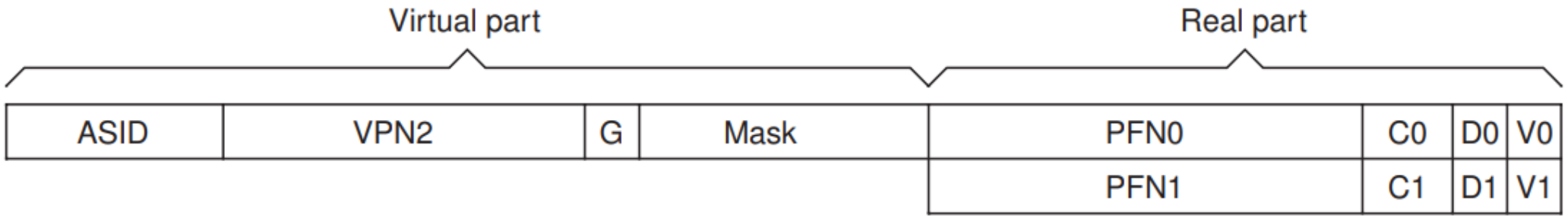


TLB-Based MIPS Memory Management

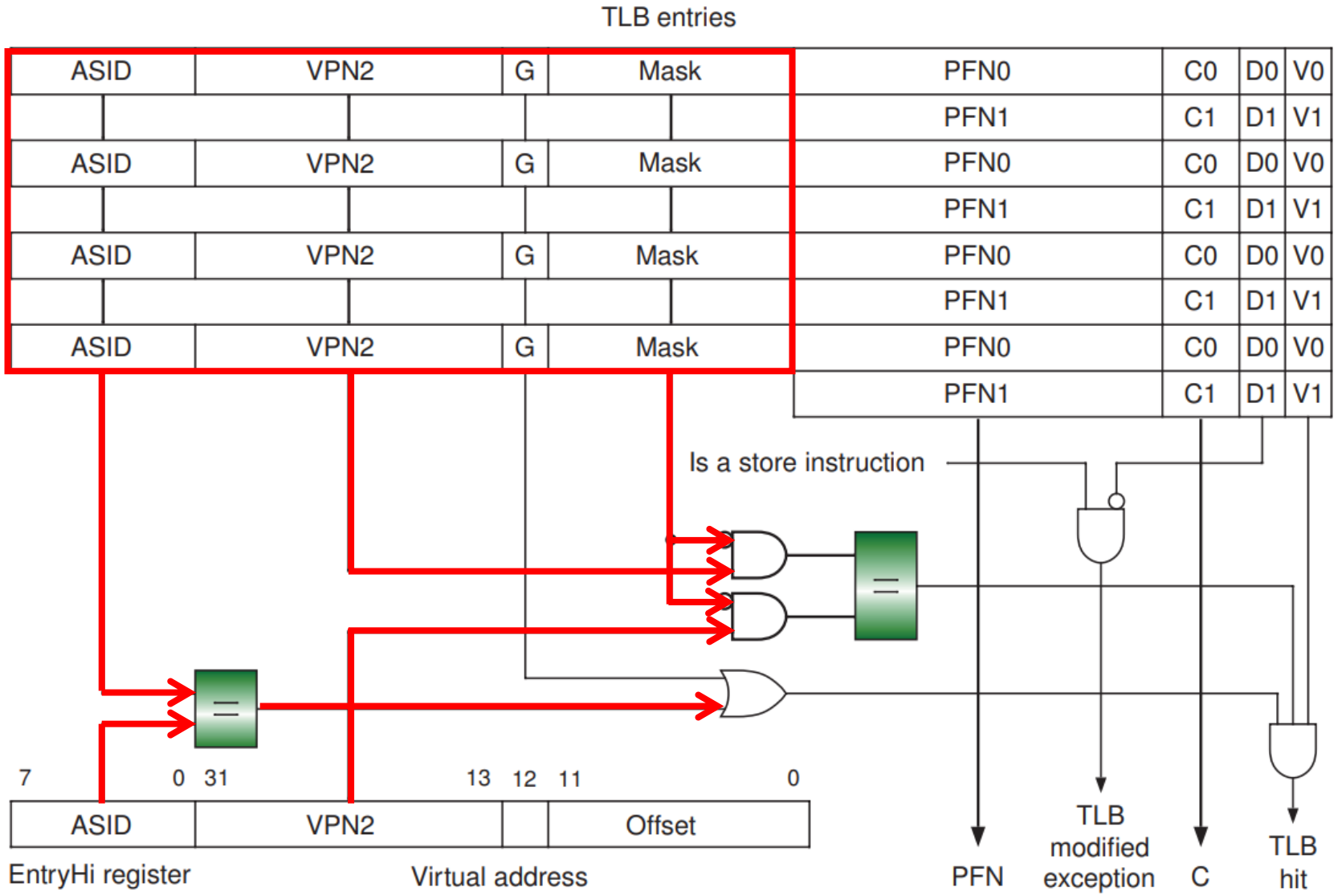
- The MIPS CPU uses a TLB-based address translation.
- MIPS CPU runs at one of two privilege modes
- User mode
  - ✓The program has access only to the general-purpose register file, floating-point register file, and up to 2GB virtual memory
- Kernal mode
  - ✓The program has access to 4GB virtual memory, and all register files include **CP0 registers**.
  - ✓Privileged instructions can be executed.

TLB-Based MIPS Memory Management

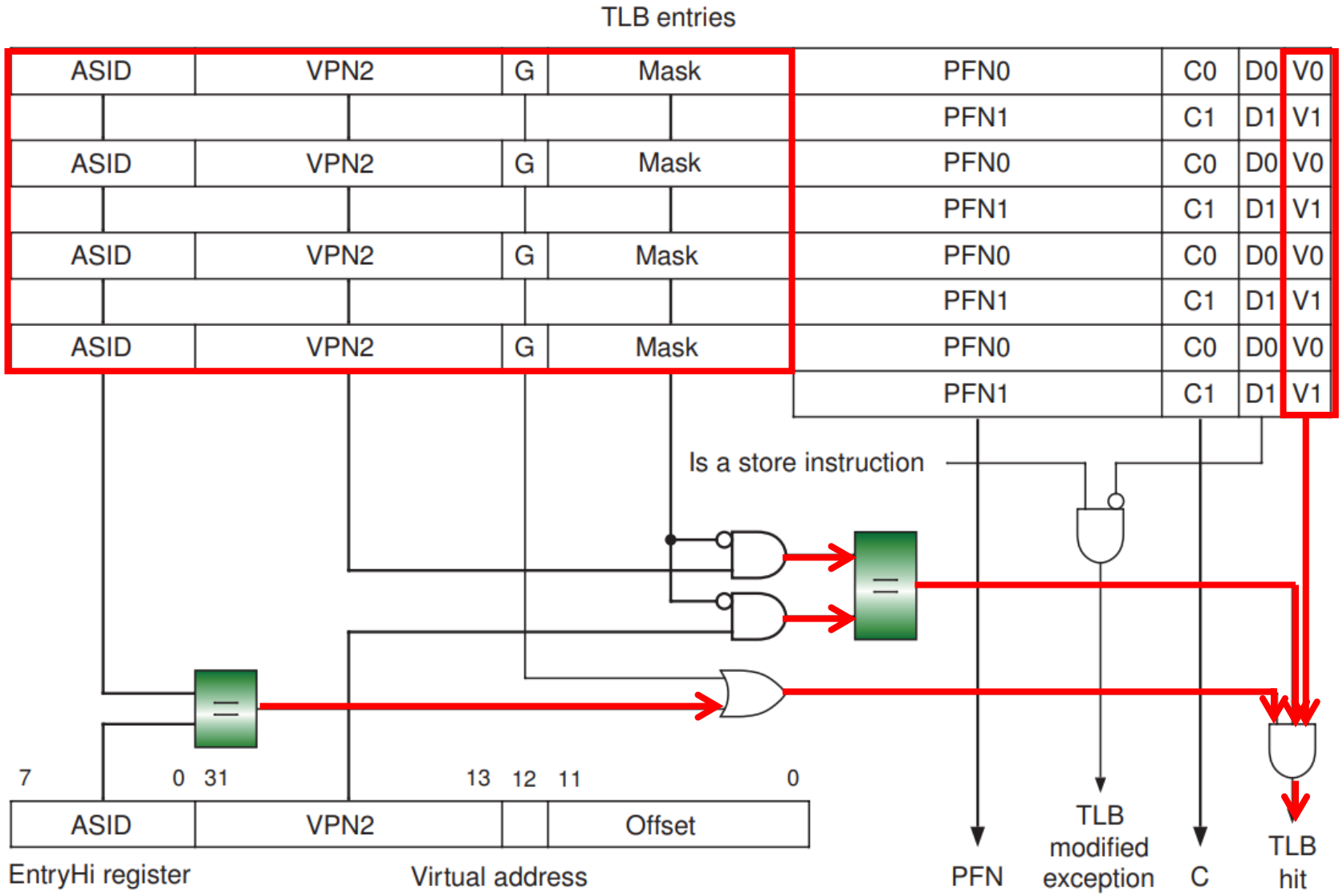
- **Organization of MIPS TLB**
- In MIPS architecture, virtual addresses are extended with an 8-bit **address space identifier (ASID)** to distinguish between processes.
- The **ASID** is stored in every TLB entry.
- Each entry contains a virtual part and a real part.
  - ✓Virtual part (ASID, VPN2, G, and Mask)
  - ✓Real part (entry 0, entry 1, PFN, C, D, and V)



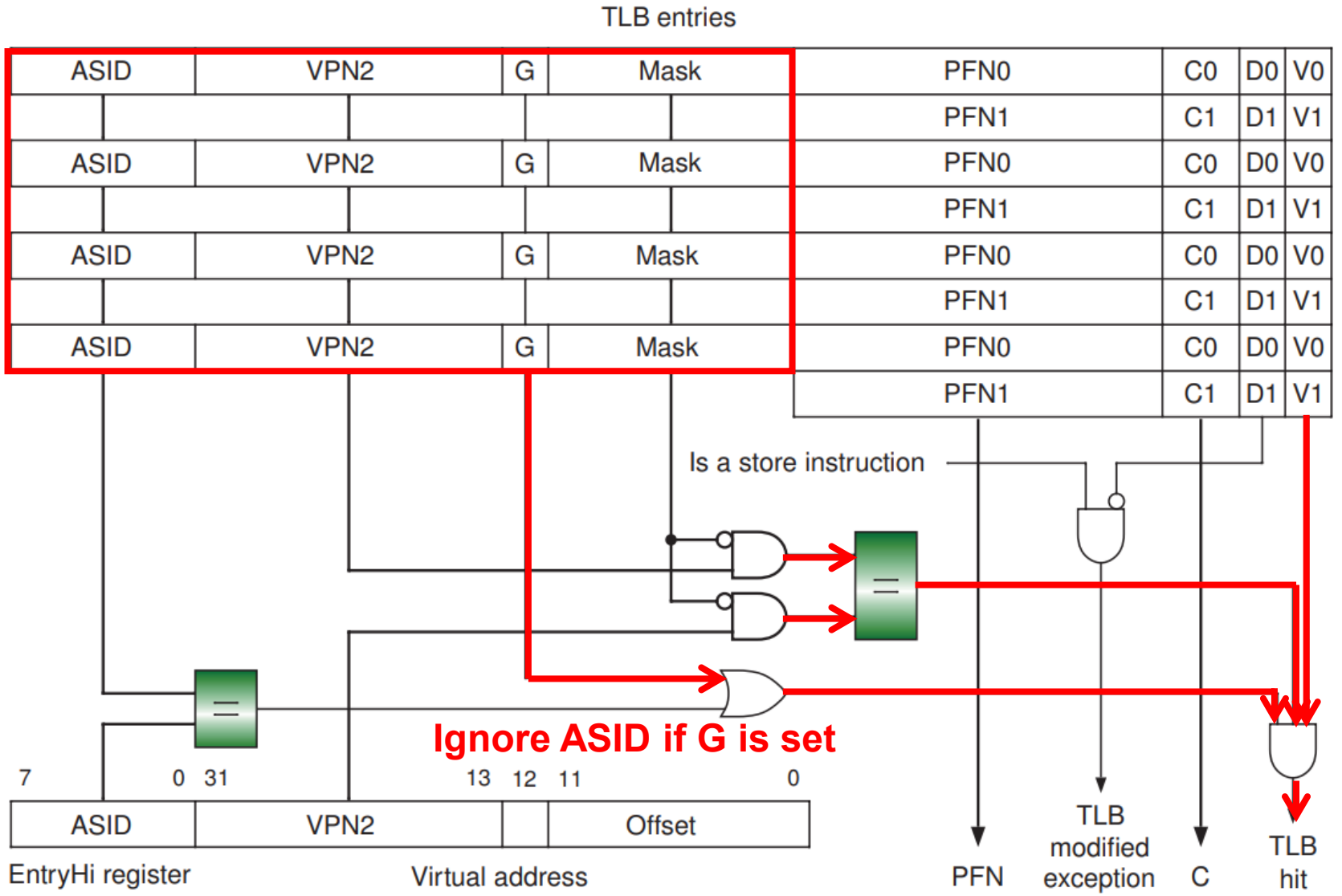
TLB-Based MIPS Memory Management



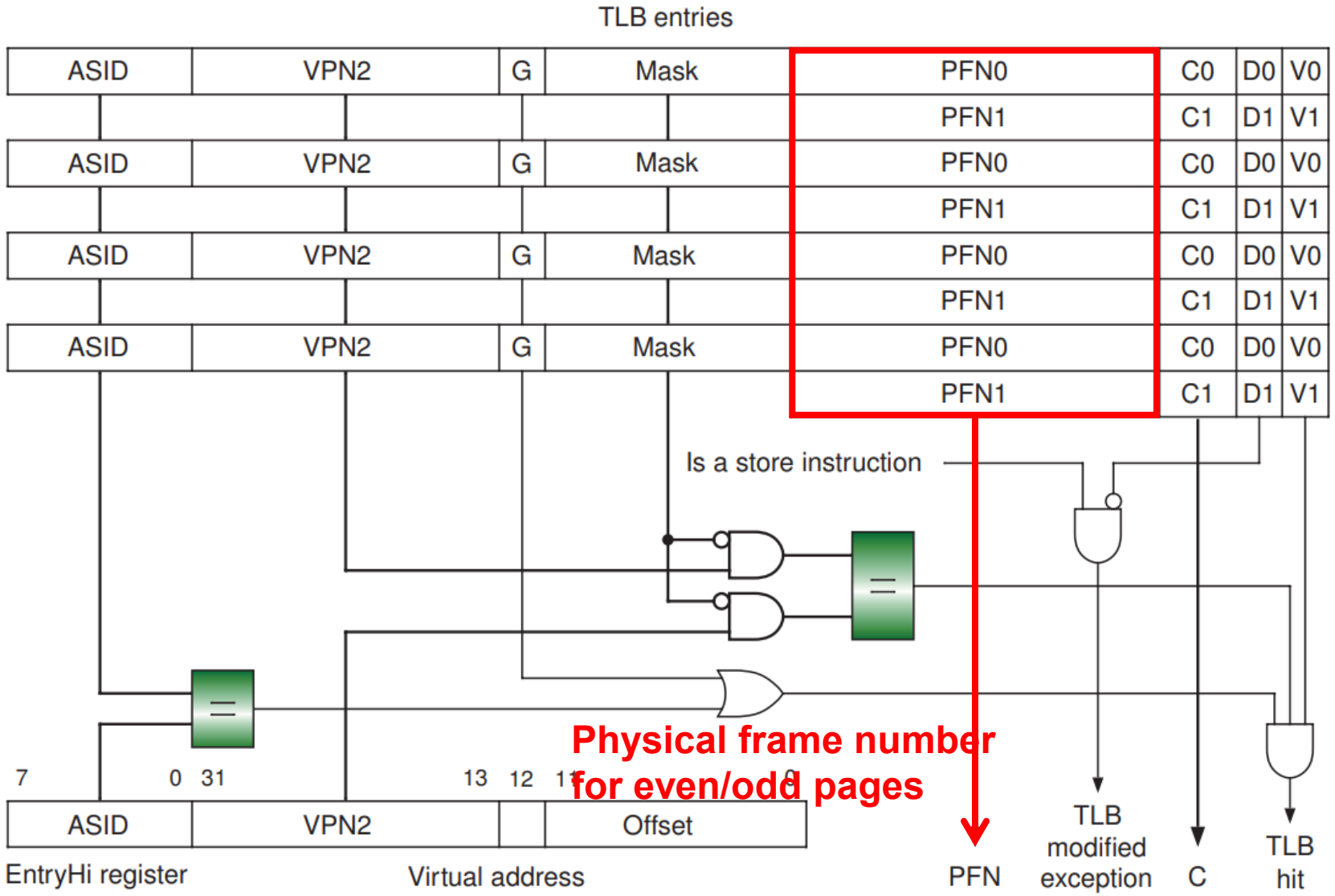
TLB-Based MIPS Memory Management



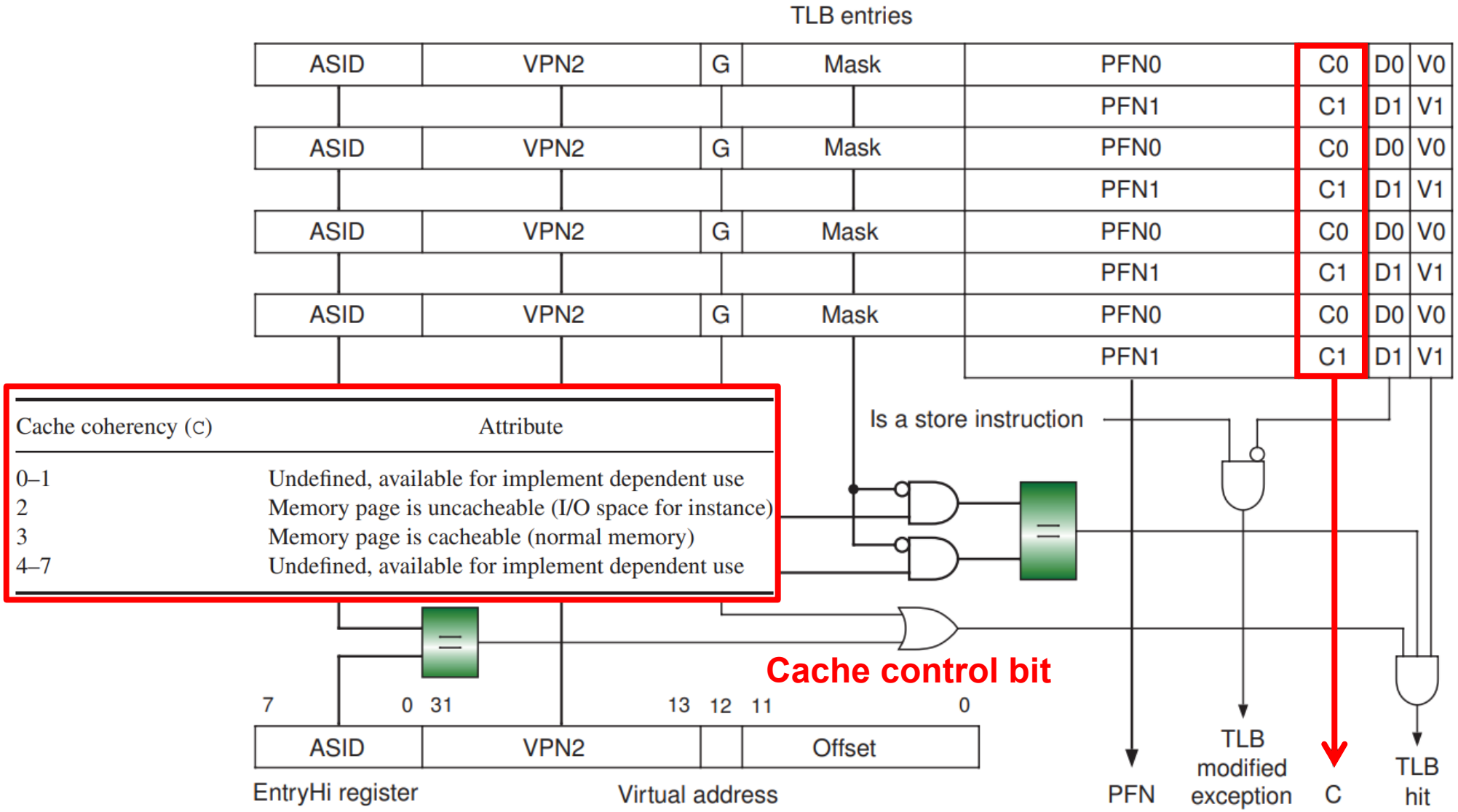
TLB-Based MIPS Memory Management



TLB-Based MIPS Memory Management

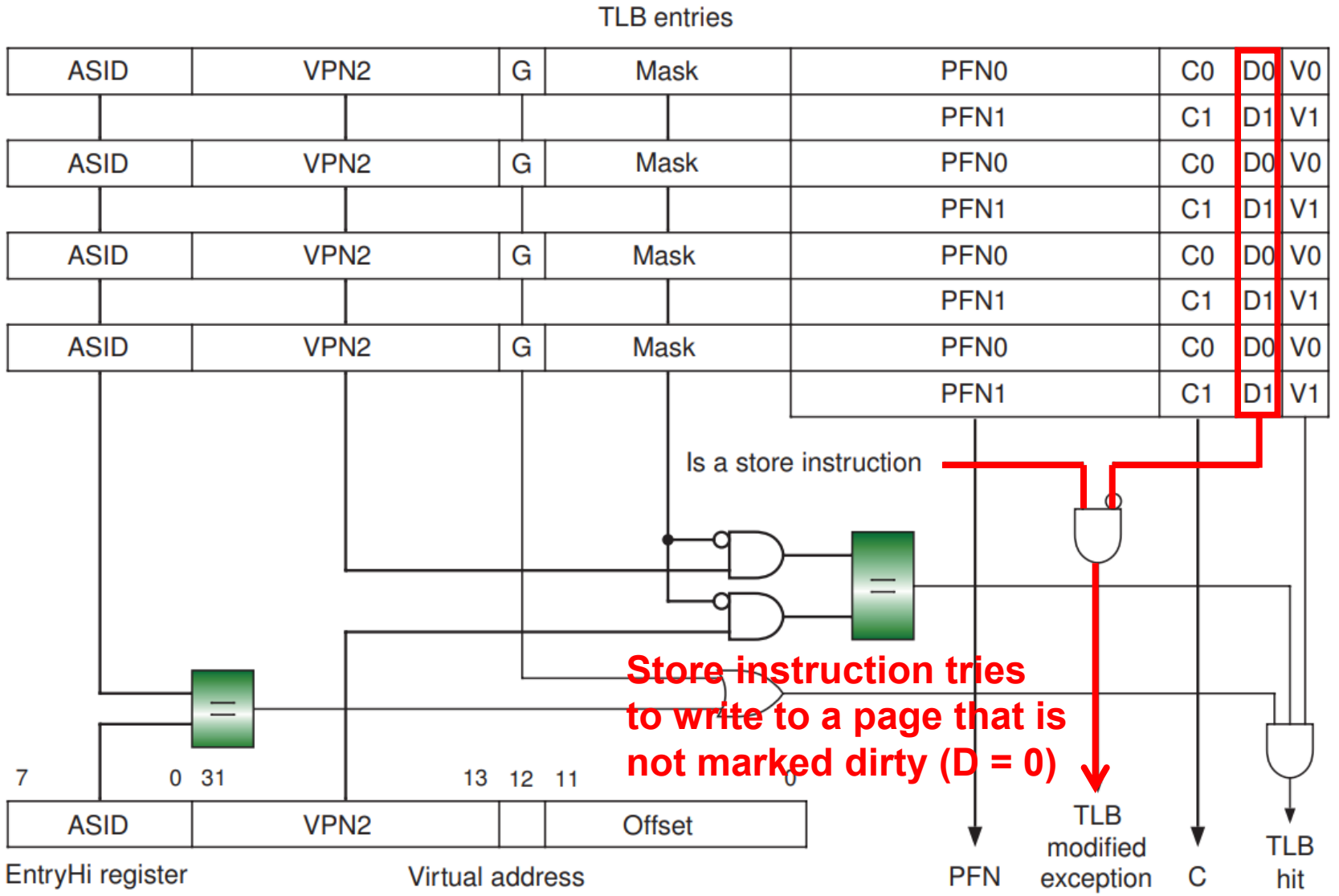


TLB-Based MIPS Memory Management



Cache coherency (C)	Attribute
0–1	Undefined, available for implement dependent use
2	Memory page is uncacheable (I/O space for instance)
3	Memory page is cacheable (normal memory)
4–7	Undefined, available for implement dependent use

TLB-Based MIPS Memory Management

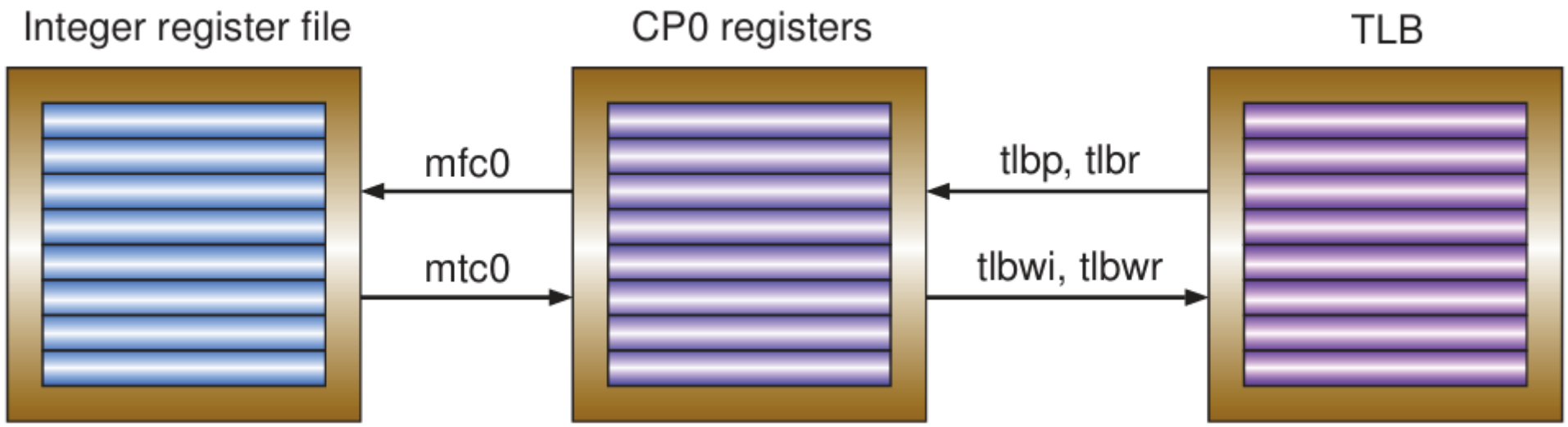


TLB-Based MIPS Memory Management

- The MIPS TLB is software-managed; there are instructions to manage the TLB's contents.
  - ✓ ***tlbp***
    - probes the TLB to see if a particular translation is in there and puts the search result on a CP0 register.
  - ✓ ***tlbr***
    - reads the contents of a TLB entry into CP0 registers.
  - ✓ ***tlbwi***
    - writes a specific TLB entry with the contents of CP0 registers.
  - ✓ ***tlbwr***
    - writes a random TLB entry with the contents of CP0 registers.
  - ✓ ***mfc0, mtc0***
    - transfer data between a CP0 register and an IU register in the general-purpose register file.

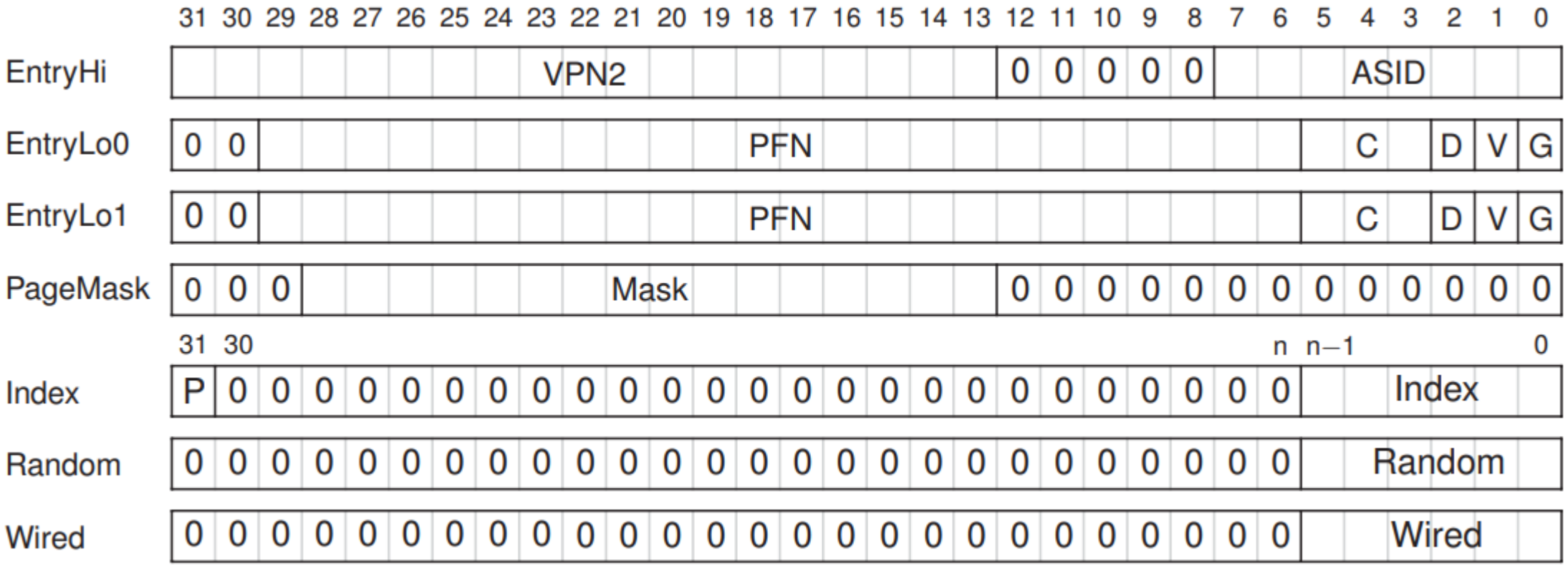
TLB-Based MIPS Memory Management

- CPU manipulating TLB through CP0 registers



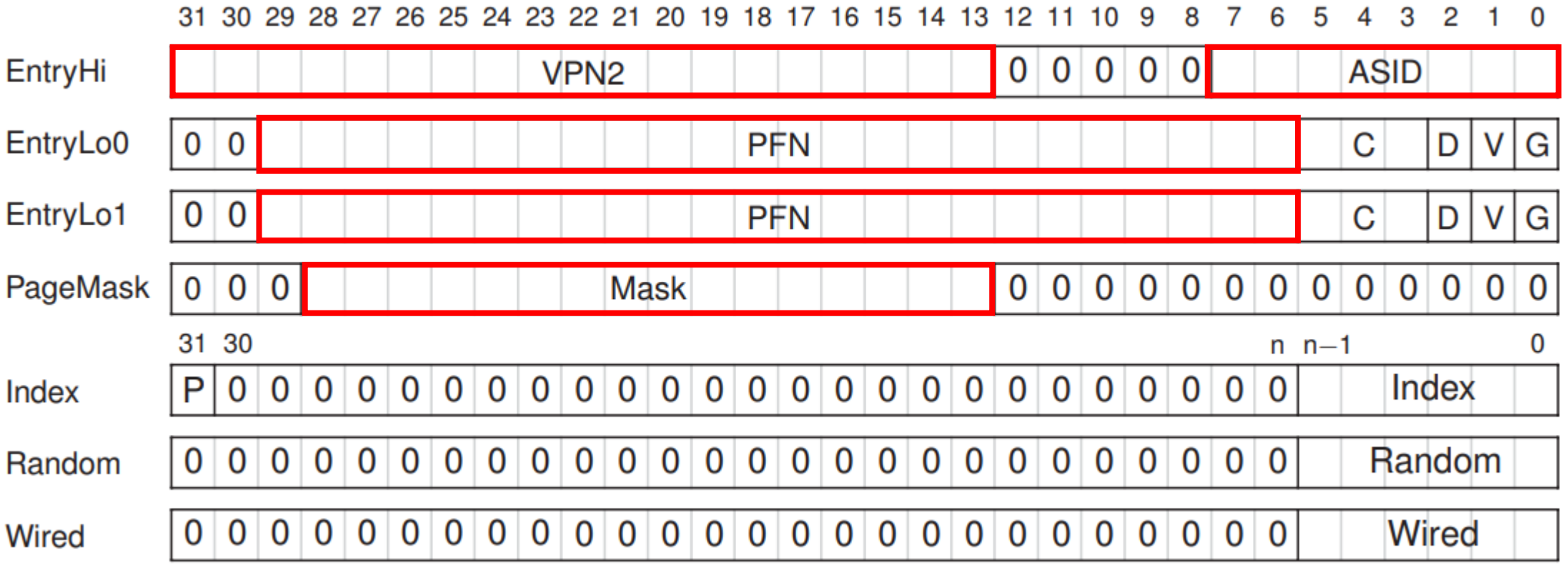
TLB-Based MIPS Memory Management

- MIPS TLB manipulation CP0 registers



TLB-Based MIPS Memory Management

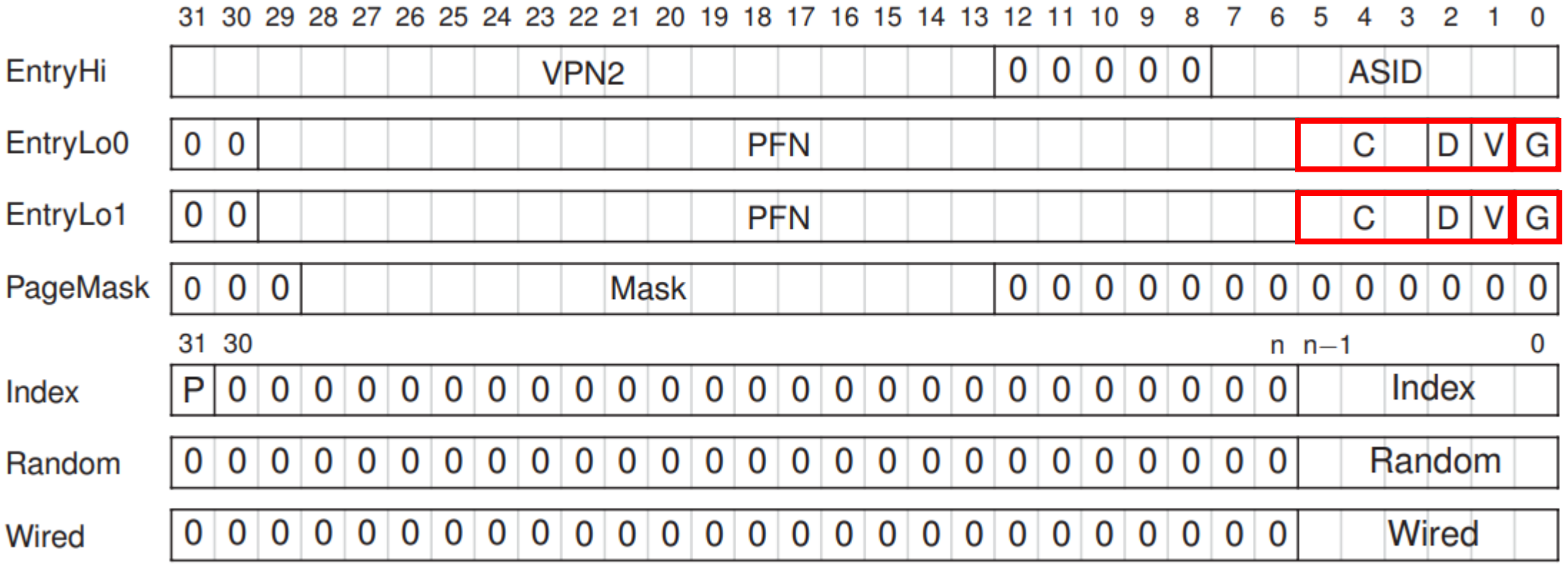
- MIPS TLB manipulation CP0 registers



Correspond exactly to the fields of the TLB entry

TLB-Based MIPS Memory Management

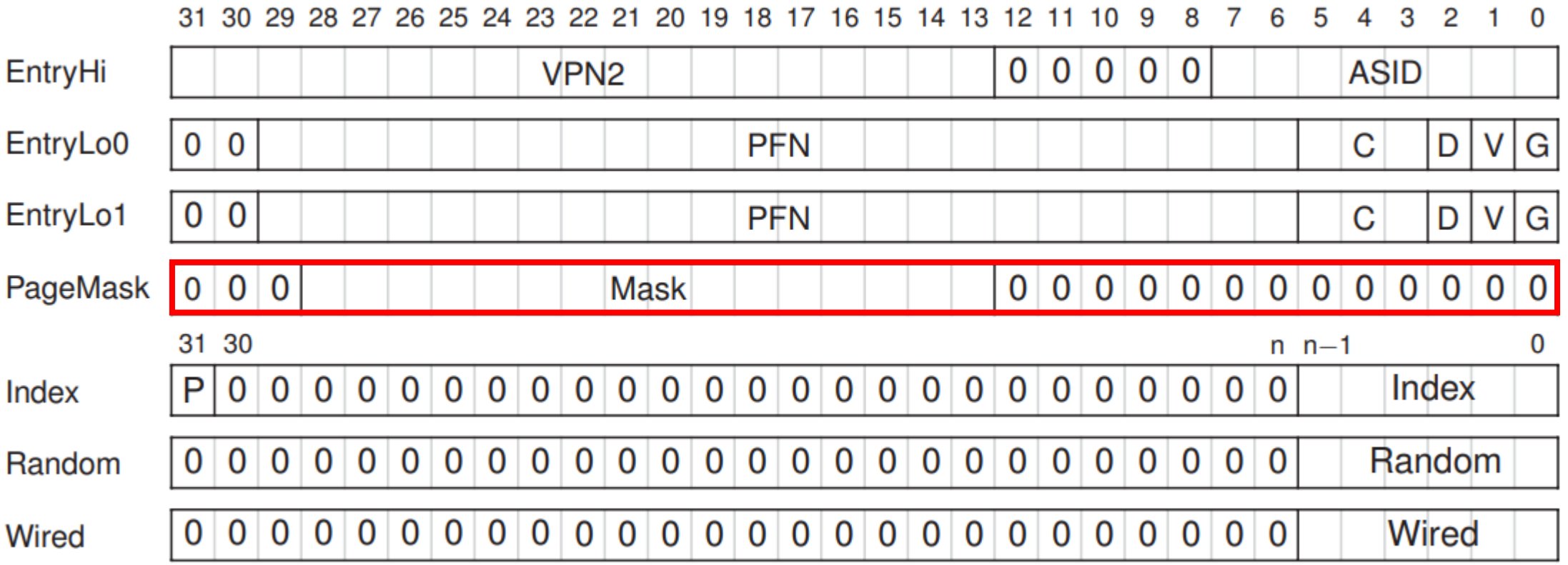
- MIPS TLB manipulation CP0 registers



Two G bits (G0 and G1)
TLB write G = G0&G1, TLB read G0=G1=G

TLB-Based MIPS Memory Management

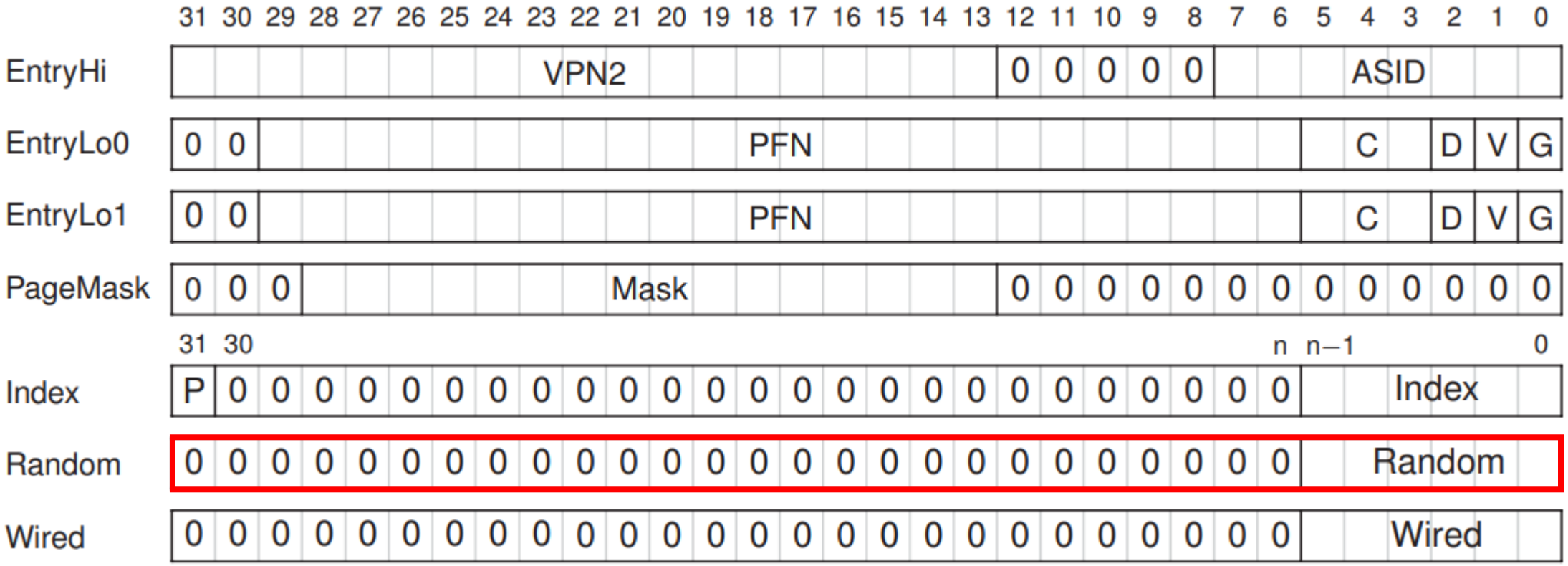
- MIPS TLB manipulation CP0 registers



PageMask register holds a mask pattern that defines the page size for each TLB entry

TLB-Based MIPS Memory Management

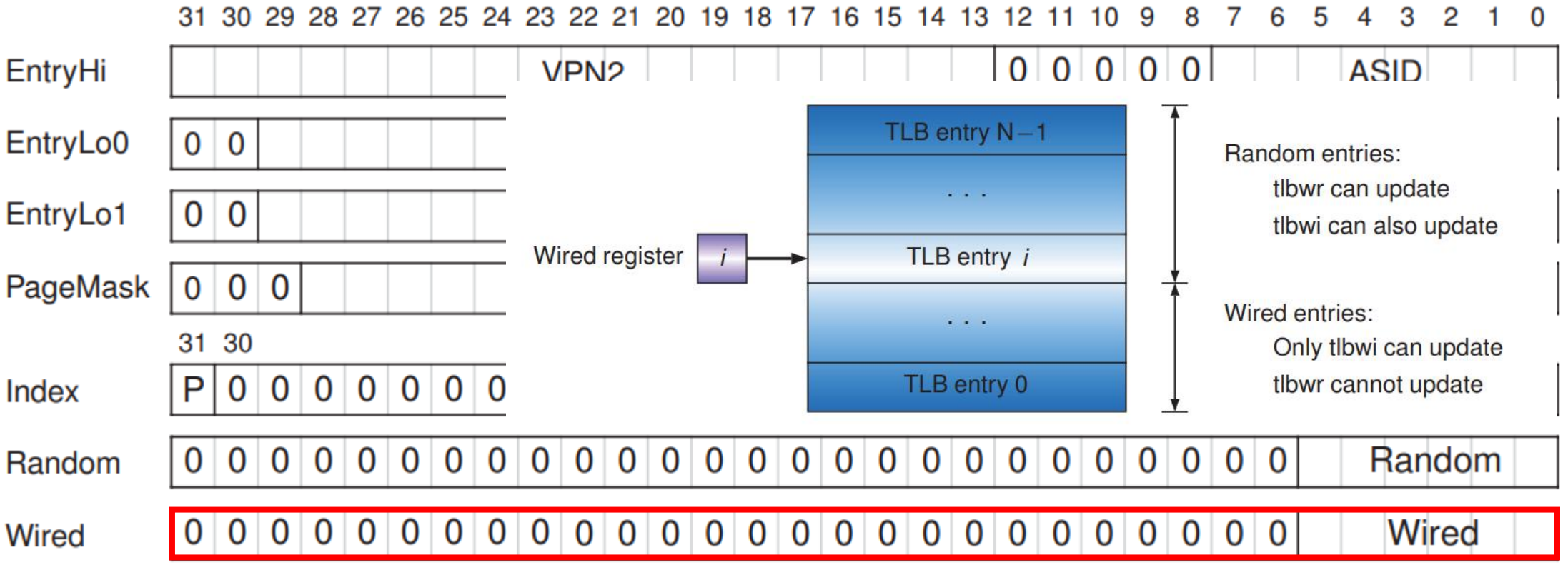
- MIPS TLB manipulation CP0 registers



Random register holds a random number generated by HW that is used to index the TLB for the *tlbwr* instruction

TLB-Based MIPS Memory Management

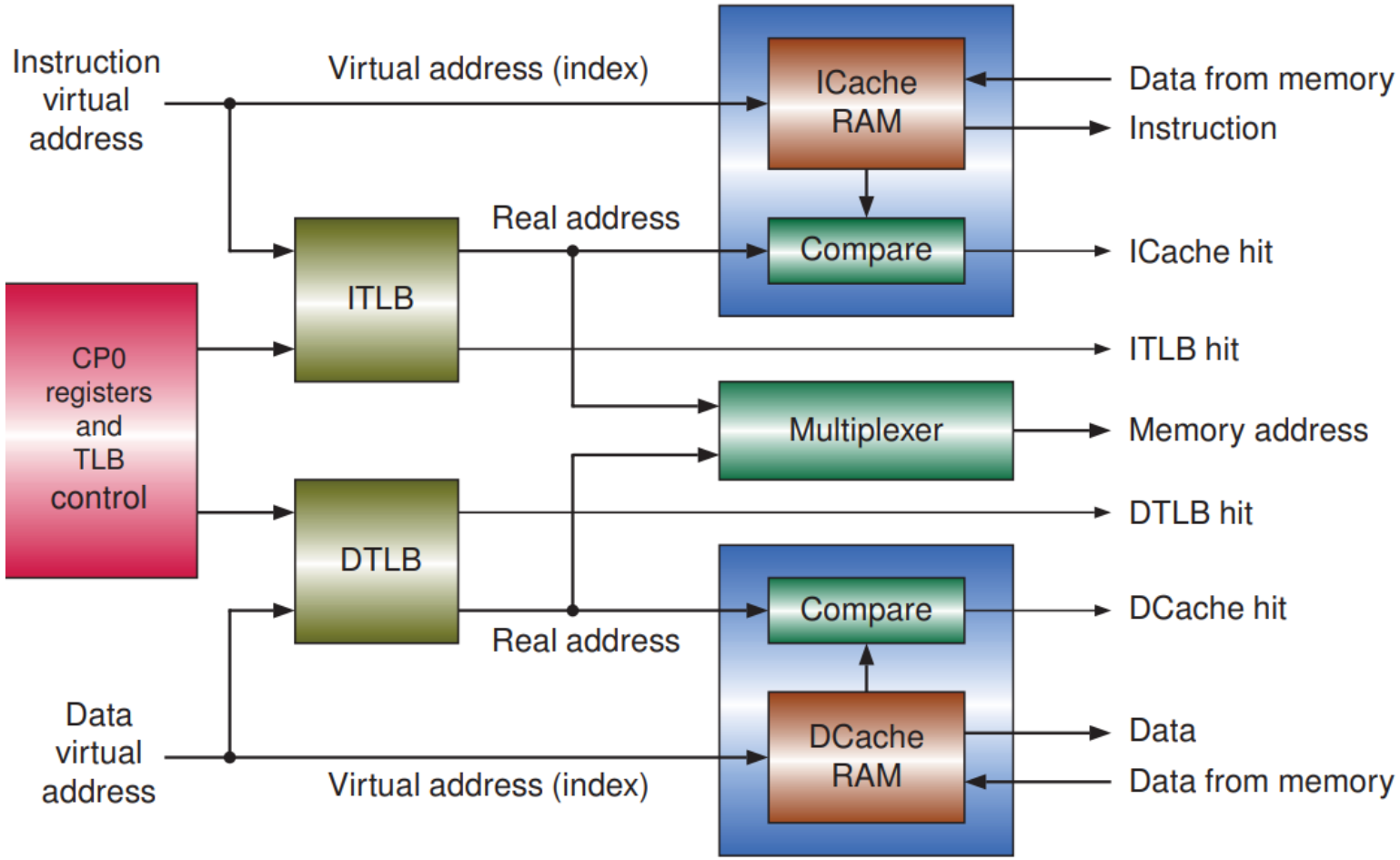
- MIPS TLB manipulation CP0 registers



Wired register is used to specify the boundary between the wired (reserved) and random entries in the TLB

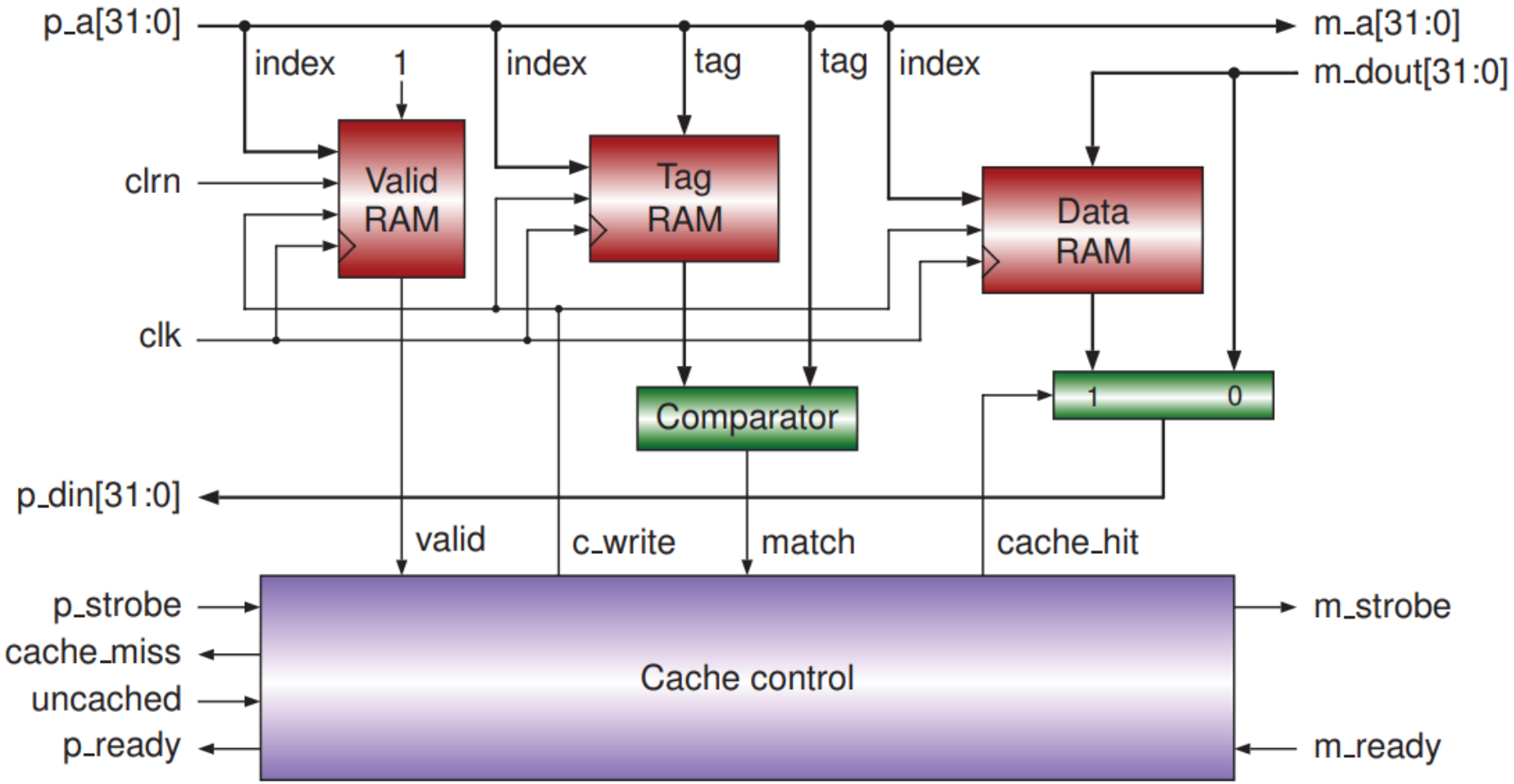
Design of Pipelined CPU with Caches and TLB

- Overall structure of caches and TLBs



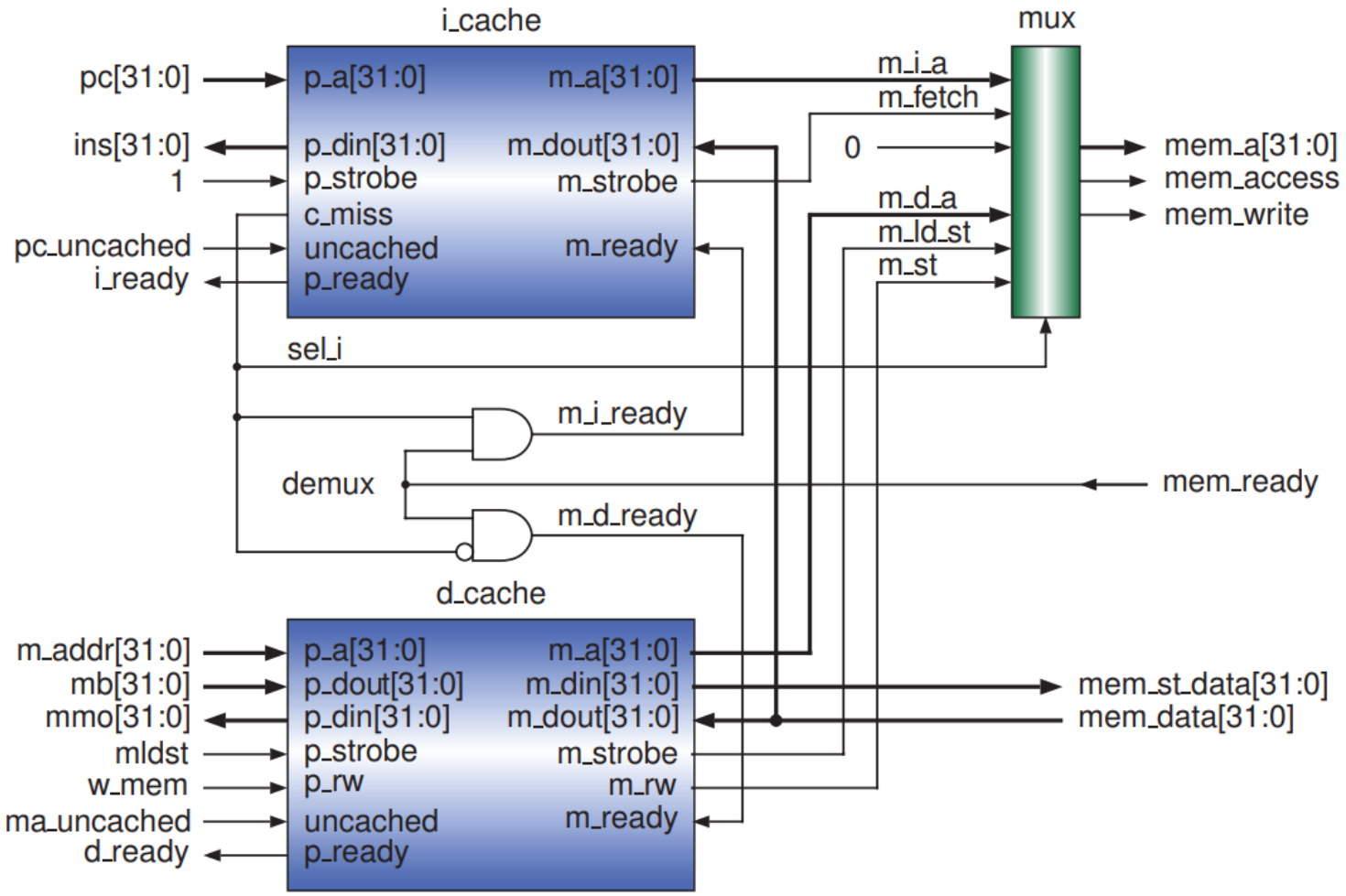
Design of Pipelined CPU with Caches and TLB

- Instruction cache



Design of Pipelined CPU with Caches and TLB

- Memory interface to instruction and data caches



Design of Pipelined CPU with Caches and TLB

- **Instruction cache**
- Processor interface
  - ✓p_a (address), p_din, p_strobe (request), and p_ready.
  - ✓cache_miss
- Memory interface
  - ✓m_a (address), m_dout, m_strobe (request), and m_ready.

```
1  module i_cache (
2      input       [31:0]  p_a,
3      output      [31:0]  p_din,
4      input               p_strobe,
5      input               uncached,
6      output              p_ready,
7      output              cache_miss,
8      input               clk, clrn,
9      output      [31:0]  m_a,
10     input       [31:0]  m_dout,
11     output              m_strobe,
12     input               m_ready
13 );
```

Design of Pipelined CPU with Caches and TLB

- **Instruction cache**
- d_valid, d_tags, d_data are indexed from 0 to 63.
  - ✓64 cache lines.
- d_data stores 32-bit instruction.
  - ✓cache line size is 1 word (32 bits).
- p_a (processor address) is dived into 3 parts.
  - ✓index (p_a[7:2]), tag (p_a[31:8]), and byte offset (p_a[1:0]).

```
14    reg             d_valid [0:63];
15    reg      [23:0] d_tags  [0:63];
16    reg      [31:0] d_data  [0:63];
17    wire     [5:0]  index   = p_a[7:2];
18    wire     [23:0] tag     = p_a[31:8];
19    wire            c_write;
20    wire     [31:0] c_din;
```

Design of Pipelined CPU with Caches and TLB

- **Instruction cache**
- d_valid is cleared when clrn is asserted.
- c_write is set, d_valid bit for index is set, indicating the valid line.

```
22      integer i;
23      always @(posedge clk or negedge clrn) begin
24          if (!clrn) begin
25              for (i=0; i<64; i=i+1) begin
26                  d_valid[i] <= 0;
27              end
28          end else if (c_write) begin
29              d_valid[index] <= 1;
30          end
31      end
```

Design of Pipelined CPU with Caches and TLB

- **Instruction cache**
- c_write is set, tag and data are written specified by index.

```
33      always @(posedge clk) begin
34          if (c_write) begin
35              d_tags[index] <= tag;
36              d_data[index] <= c_din;
37          end
38      end
```

Design of Pipelined CPU with Caches and TLB

- **Instruction cache**
- cache_hit: p_strobe is set, valid index, and tag match.
- cache_miss: p_strobe is set but invalid index or tag not match.
- m_strobe: when cache_miss, cache asserts request to main memory at the m_a (= p_a).
- p_ready: cache_hit or cache_miss and m_ready (main memory).
- c_write: cache_miss, not uncached, and m_ready.
- p_din: c_dout if cahche_hit or m_dout if cache_miss.

```
40    wire               valid   = d_valid[index];
41    wire       [23:0]  tagout  = d_tags[index];
42    wire       [31:0]  c_dout  = d_data[index];
43    wire               cache_hit   = p_strobe & valid & (tagout == tag);
44    assign             cache_miss  = p_strobe & (!valid | (tagout != tag));
45    assign             m_a         = p_a;
46    assign             m_strobe    = cache_miss;
47    assign             p_ready     = cache_hit | cache_miss & m_ready;
48    assign             c_write     = cache_miss & ~uncahced & m_ready;
49    assign             c_din       = m_dout;
50    assign             p_din       = cache_hit ? c_dout : m_dout;
```

Design of Pipelined CPU with Caches and TLB

- **Data cache**

```
1  module d_cache (    // direct mapping, 2^6 blocks, 1 world/block, write through
2      input   [31:0]  p_a,                    // cpu address
3      input   [31:0]  p_dout,                 // cpu data out to mem
4      output  [31:0]  p_din,                  // cpu data in from mem
5      input           p_storbe,               // cpu strobe
6      input           p_rw,                   // cpu read/write command
7      input           uncached,               // uncached
8      output          p_ready,                // ready (to cpu)
9      input           clk, clrn,              // clock and reset
10     output  [31:0]  m_a,                    // mem address
11     input   [31:0]  m_dout,                 // mem data out to cpu
12     output  [31:0]  m_din,                  // mem data in from cpu
13     output          m_strobe,               // mem strobe
14     output          m_rw,                   // mem read/write
15     input           m_ready                 // mem ready
16 );
17     reg             d_valid [0:63];         // 1-bit valid
18     reg     [23:0]  d_tags  [0:63];         // 24-bit tag
19     reg     [31:0]  d_data  [0:63];         // 32-bit data
20     wire    [23:0]  tag = p_a[31:8];        // address tag
21     wire    [31:0]  c_din;                  // data to cache
22     wire    [5:0]   index = p_a[7:2];       // block index
23     wire            c_write;                // cache write
```

Design of Pipelined CPU with Caches and TLB

- **Data cache**
- d_valid is cleared when clrn is asserted.
- c_write is set, valid, tags, and data are written.

```
25      integer i;
26      always @(posedge clk or negedge clrn) begin
27          if (!clrn) begin
28              for (i=0; i<64; i=i+1) begin
29                  d_valid[i] <= 0;
30              end
31          end else if (c_write) begin
32              d_valid[index] <= 1;
33          end
34      end
35
36      always @(posedge clk) begin
37          if (c_write) begin
38              d_tags[index] <= tag;
39              d_data[index] <= c_din;
40          end
41      end
```

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

Design of Pipelined CPU with Caches and TLB

- **Data cache**
- cache_hit: p_strobe is set, valid index, and tag match.
- cache_miss: p_strobe is set but invalid index or tag not match.

```
43      wire            valid   = d_valid[index];
44      wire    [23:0]  tagout  = d_tags[index];
45      wire    [31:0]  c_dout  = d_data[index];
46      wire    cache_hit   = p_strobe & valid & (tagout == tag);
47      wire    cache_miss  = p_strobe & ( !valid | (tagout != tag) );
```

Design of Pipelined CPU with Caches and TLB

- **Data cache**
- m_din: directly passes p_dout (CPU's write data) to memory.
- m_a: directly passes p_a (CPU's address) to memory.
- m_rw: directly passes p_rw (CPU's rd/wr command) to memory.
- m_strobe: Main memory is strobes for a transaction if
  - ✓p_rw is set, because of the write-through policy.
  - ✓or a cache miss

```
48      assign  m_din       = p_dout;
49      assign  m_a         = p_a;
50      assign  m_rw        = p_rw;
51      assign  m_strobe    = p_rw | cache_miss;
```

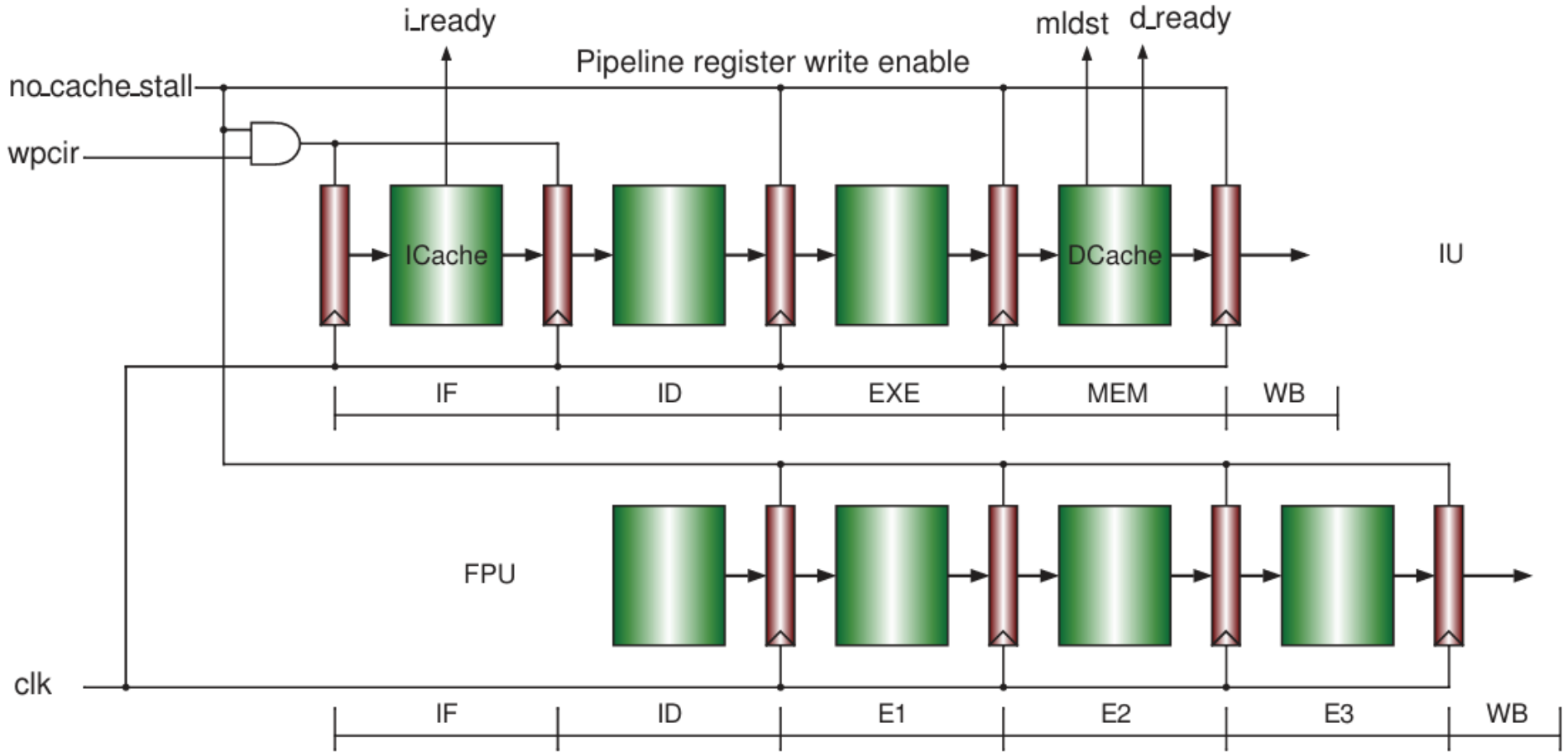
Design of Pipelined CPU with Caches and TLB

- **Data cache**
- For reads (~p_rw),
  - ✓p_ready is asserted on a cache hit.
- For writes (p_rw) or read with miss,
  - ✓p_ready is asserted when m_ready is set (main memory).
- c_write
  - ✓uncached, CPU write (p_rw) or miss has been resolved.
- c_din
  - ✓For writing (p_rw), p_dout (the data from the CPU).
  - ✓For reading (~p_rw), m_dout (the data from the main memory).

```
52    assign  p_ready     = ~p_rw & cache_hit | (cache_miss | p_rw) & m_ready;
53    assign  c_write     = ~uncached & (p_rw | cache_miss & m_ready);
54    assign  c_din       = p_rw      ? p_dout : m_dout;
55    assign  p_din       = cache_hit ? c_dout : m_dout;
```

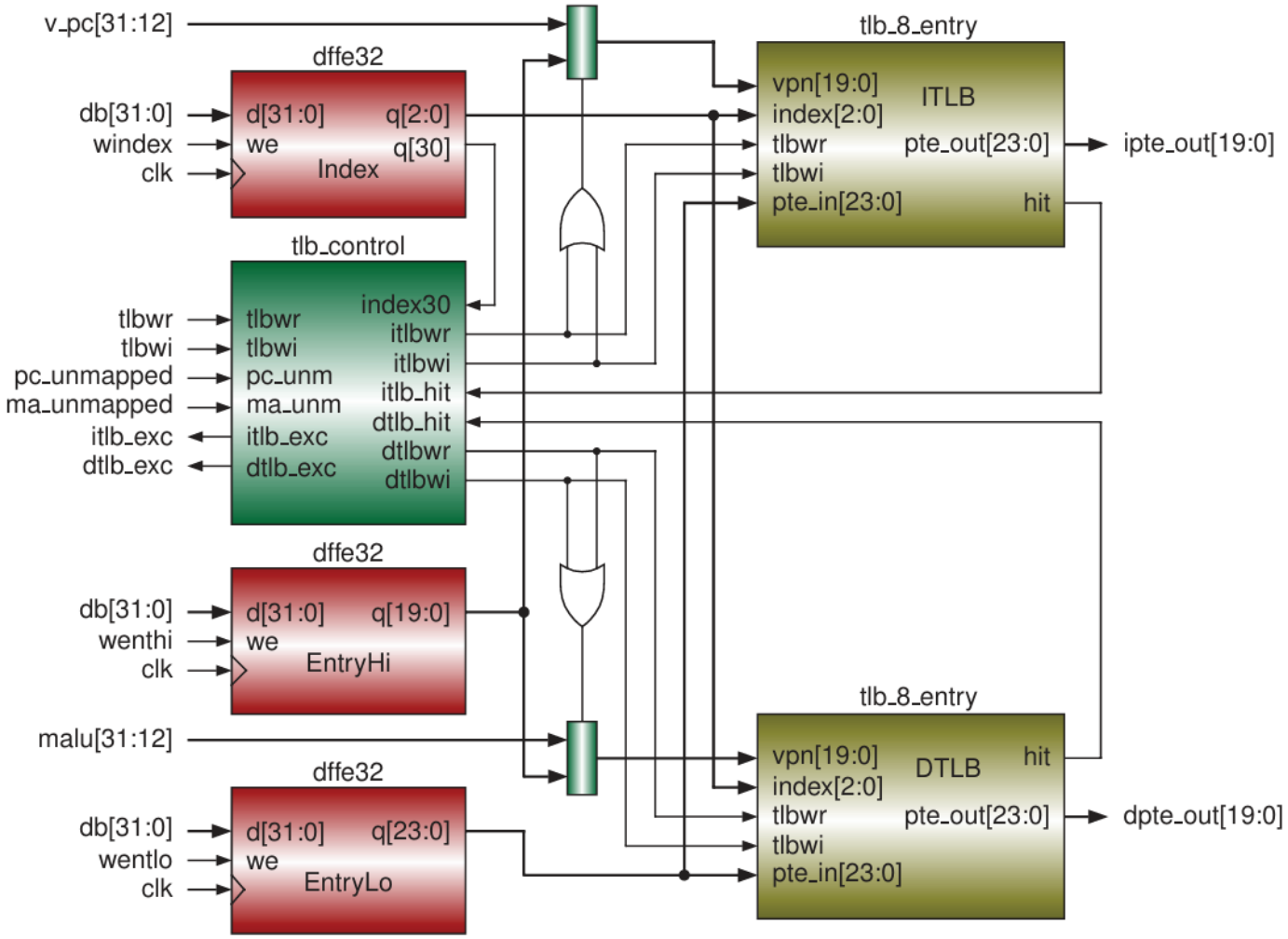
Design of Pipelined CPU with Caches and TLB

- Pipeline halt circuit for cache misses



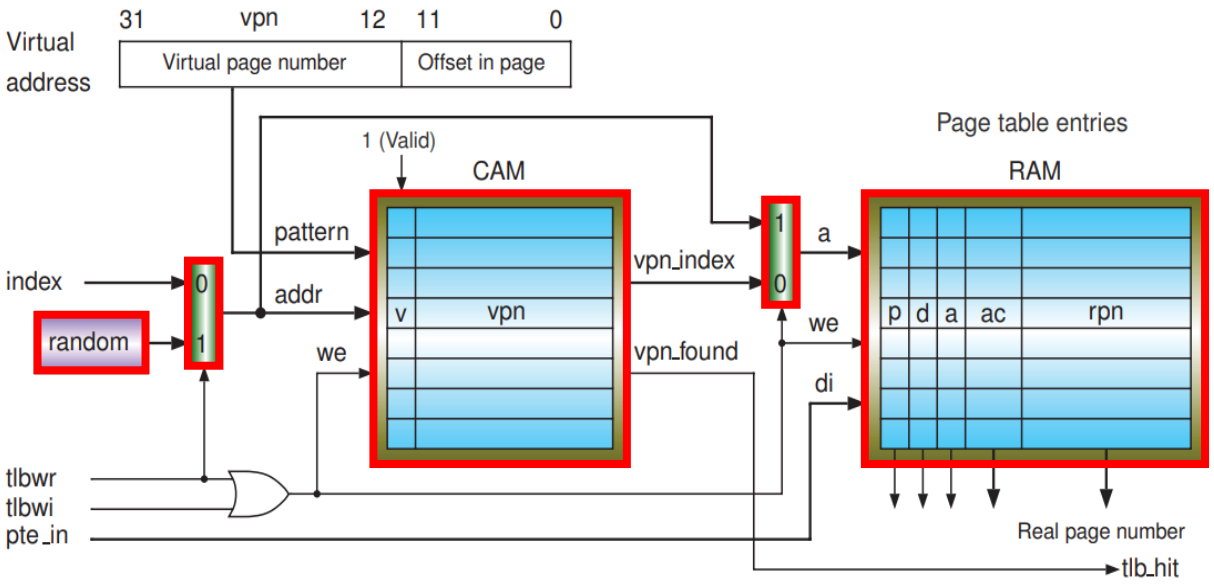
Design of Pipelined CPU with Caches and TLB

- ITLB and DTLB



Design of Pipelined CPU with Caches and TLB

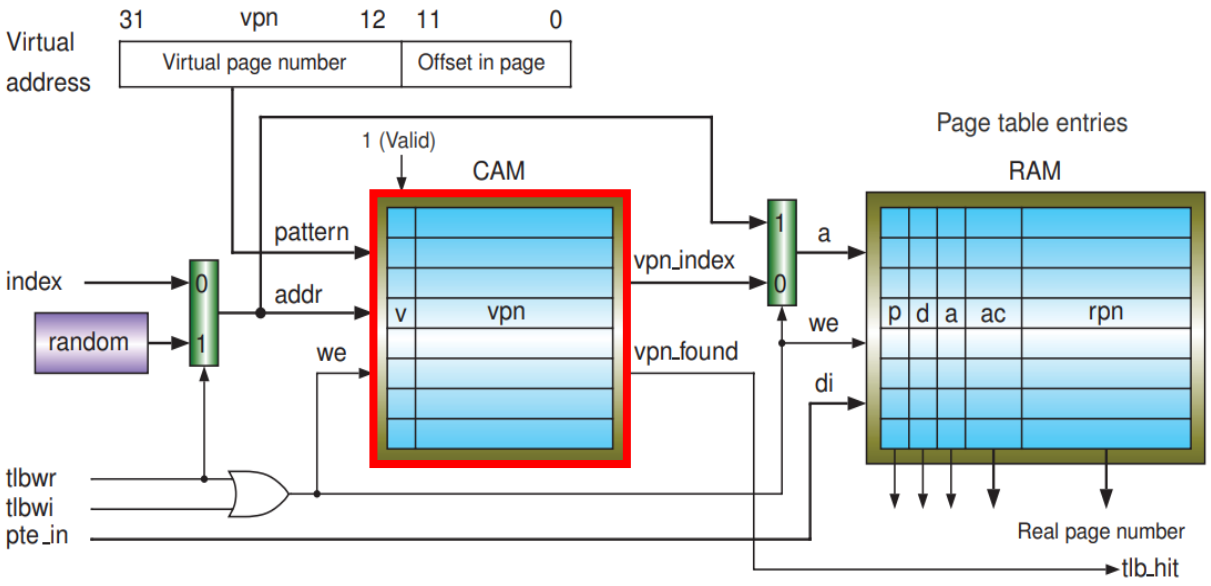
- ITLB and DTLB




```
1  module tlb_8_entry (
2      input       [23:0]  pte_in,
3      input               tlbwi,
4      input               tlbwr,
5      input       [2:0]   index,
6      input       [19:0]  vpn,
7      input               clk, clrn,
8      output      [23:0]  pte_out,
9      output              tlb_hit
10 );
11     wire        [2:0]   random;
12     wire        [2:0]   w_idx;
13     wire        [2:0]   ram_idx;
14     wire        [2:0]   vpn_index;
15     wire                tlbw    = tlbwi | tlbwr;
16     rand3   rdm (clk,clrn,random);
17     mux2x3  w_address (index,random,tlbwr,w_idx);
18     mux2x3  ram_address (vpn_index,w_idx,tlbw,ram_idx);
19     ram8x24 rpn (ram_idx,pte_in,clk,tlbw,pte_out);
20     cam8x21 valid_tag (clk,vpn,w_idx,tlbw,vpn_index,tlb_hit);
21 endmodule
```

Design of Pipelined CPU with Caches and TLB

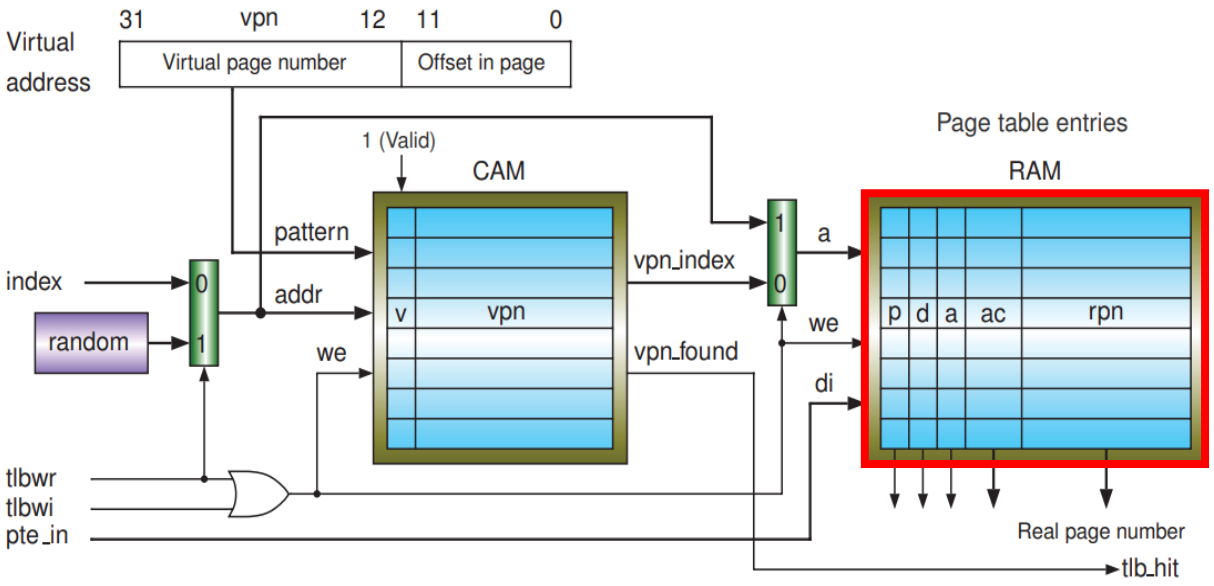
- CAM




```
module cam8x21 (
    input               clk,
    input               wren,
    input       [19:0]  pattern,
    input       [2:0]   wraddress,
    output      [2:0]   maddress,
    output              mfound
);
    // write cam, update a line with pattern, valid bit <- 1
    reg         [20:0]  ram [0:7];
    always @(posedge clk) begin
        if (wren) ram[waddress] <= {1'b1,pattern};
    end

    // fully associative search, should be implemented with CAM cells
    wire        [7:0]   match_line;
    assign match_line[7] = (ram[7] == {1'b1,pattern});
    assign match_line[6] = (ram[6] == {1'b1,pattern});
    assign match_line[5] = (ram[5] == {1'b1,pattern});
    assign match_line[4] = (ram[4] == {1'b1,pattern});
    assign match_line[3] = (ram[3] == {1'b1,pattern});
    assign match_line[2] = (ram[2] == {1'b1,pattern});
    assign match_line[1] = (ram[1] == {1'b1,pattern});
    assign match_line[0] = (ram[0] == {1'b1,pattern});
    assign mfound        = |match_line;

    // encoder for matched address, no multiple-match is allowed
    assign maddress[2] = match_line[7] | match_line[6] |
                         match_line[5] | match_line[4];
    assign maddress[1] = match_line[7] | match_line[6] |
                         match_line[3] | match_line[2];
    assign maddress[0] = match_line[7] | match_line[5] |
                         match_line[3] | match_line[1];

    // initialize cam, mainly clear valid bit of each line
    integer i;
    initial begin
        for (i=0; i<8; i=i+1)
            ram[i] = 0;
    end
endmodule
```

Design of Pipelined CPU with Caches and TLB

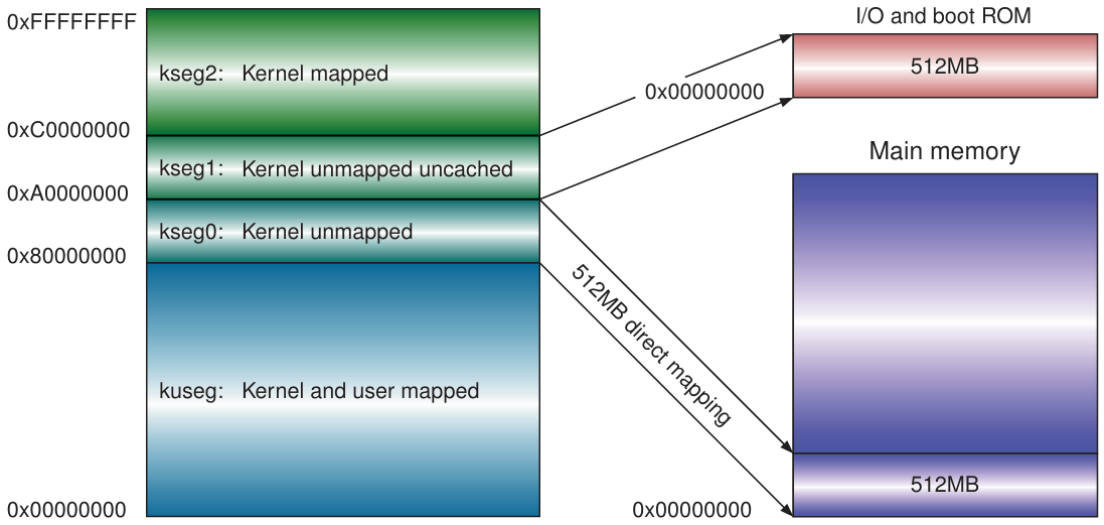
- RAM




```
1  module ram8x24 (
2      input       [2:0]   address,
3      input       [23:0]  data,
4      input               clk,
5      input               we,
6      output      [23:0]  q
7  );
8      reg         [23:0]  ram [0:7];
9      always @(posedge clk) begin
10         if (we) ram[address] <= data;
11     end
12     assign q = ram[address];
13
14     integer i;
15     initial begin
16         for (i=0 ; i<8; i=i+1)
17             ram[i] = 24'h0;
18     end
19 endmodule
```

Design of Pipelined CPU with Caches and TLB

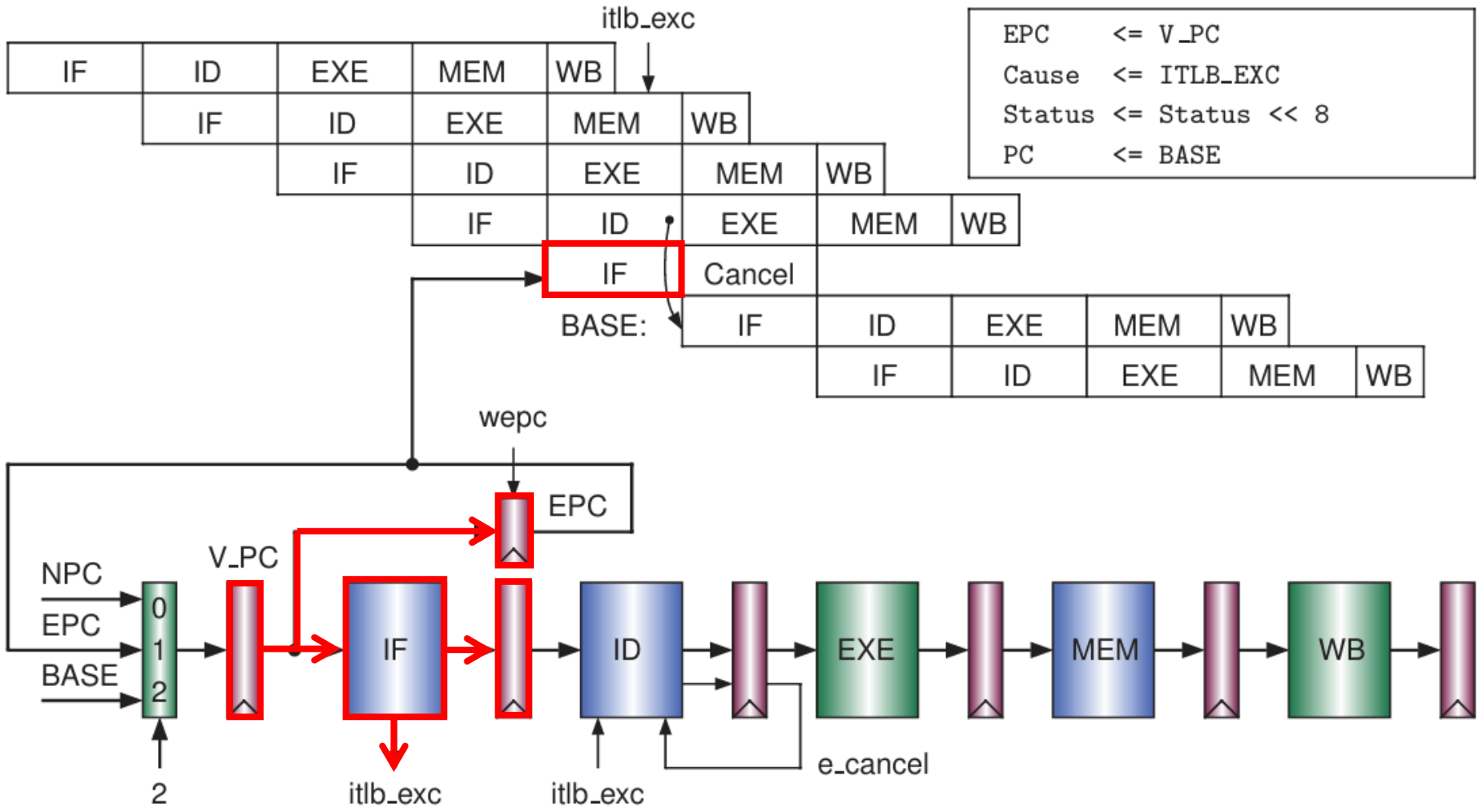
- Address mapping of the MIPS architecture



Segment	Virtual address	Physical address	TLBed	Cached	I/O
kuseg	00000000 … 0000 01111111 … 1111	00100000 … 0000 –	Yes	Yes	No
kseg0	10000000 … 0000 10011111 … 1111	00000000 … 0000 00011111 … 1111	No	Yes	No
kseg1	10100000 … 0000 10111111 … 1111	00000000 … 0000 00011111 … 1111	No	No	Yes
kseg2	11000000 … 0000 11111111 … 1111	00100000 … 0000 –	Yes	Yes	No

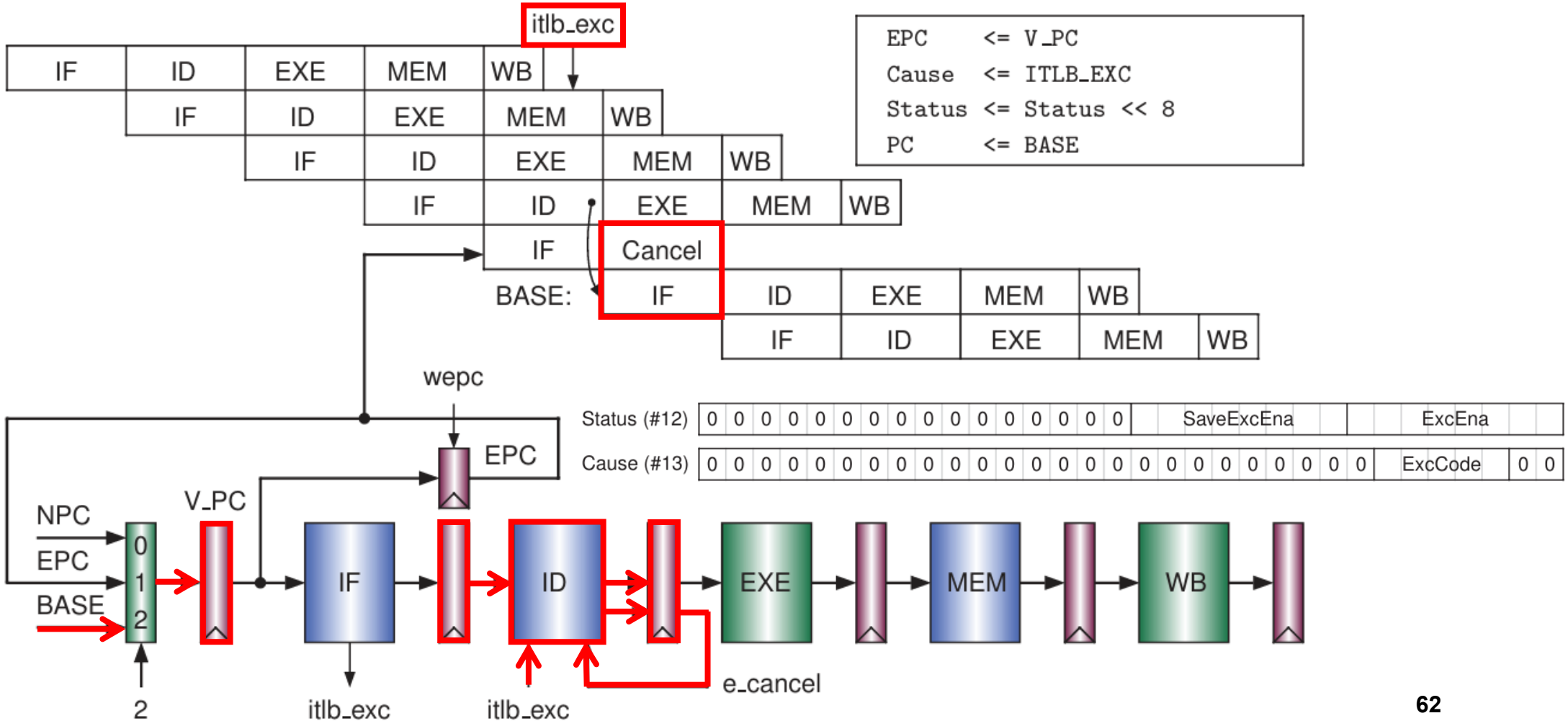
Design of Pipelined CPU with Caches and TLB

- Registers related to ITLB miss exceptions



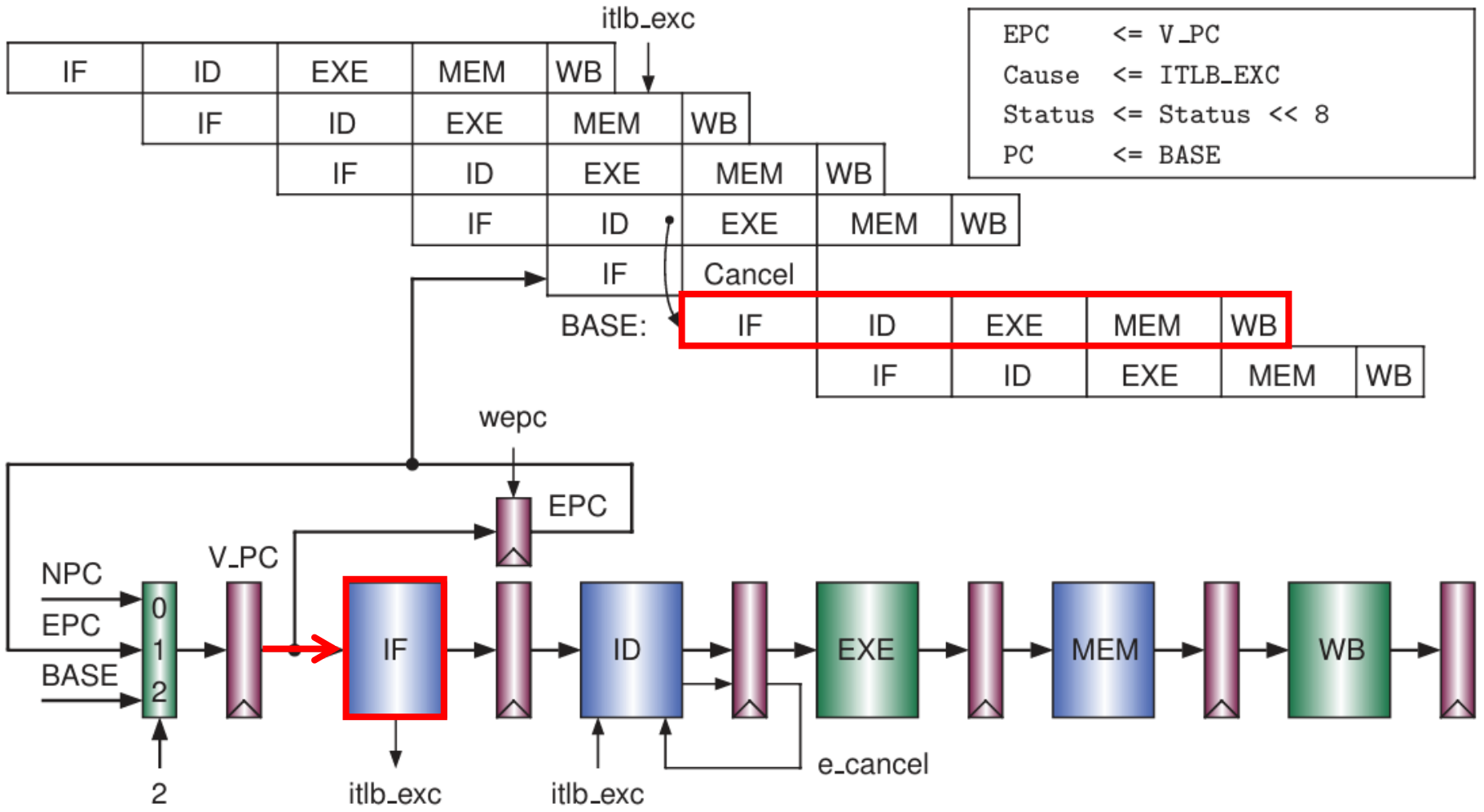
Design of Pipelined CPU with Caches and TLB

- Registers related to ITLB miss exceptions



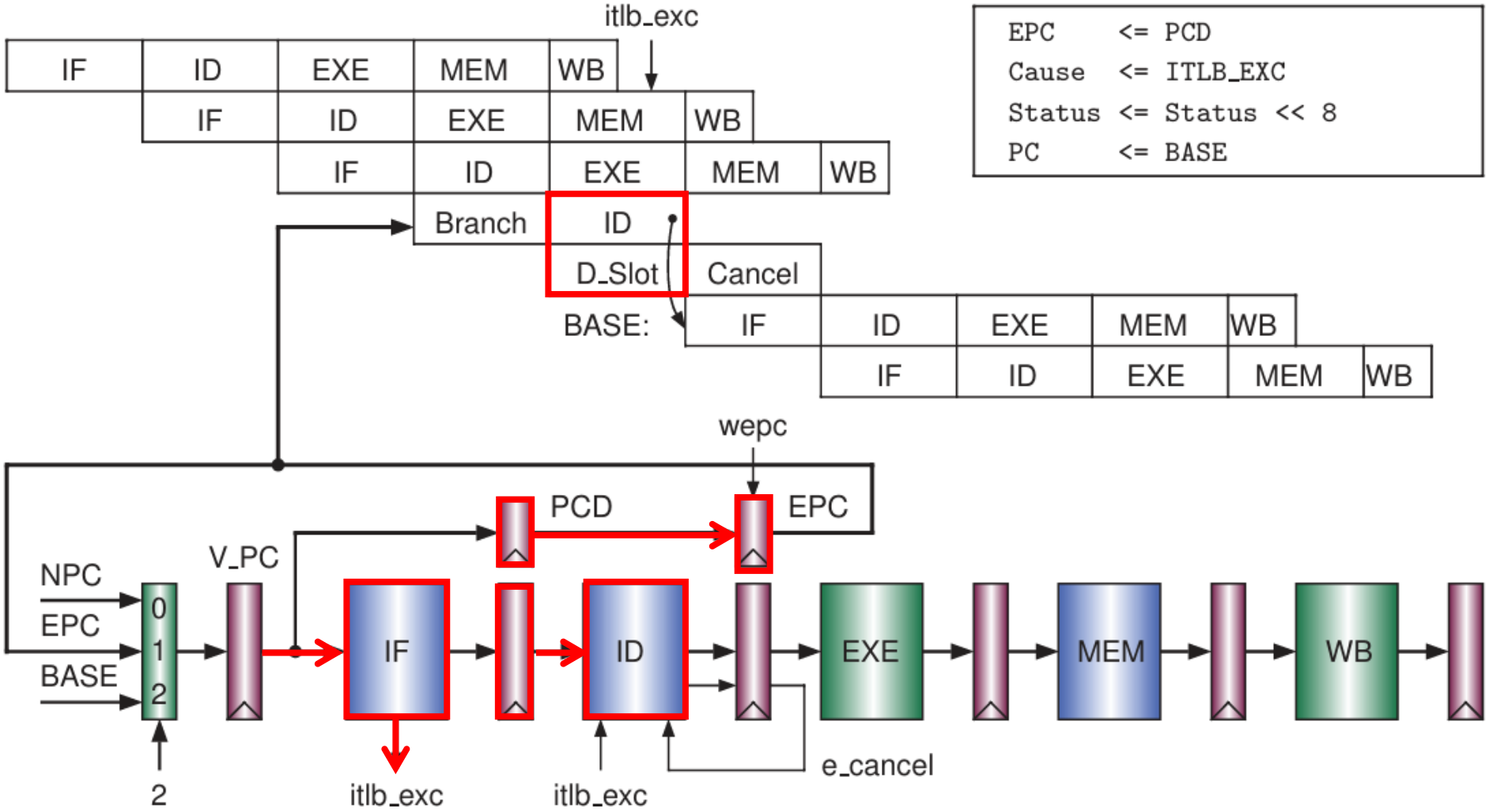
Design of Pipelined CPU with Caches and TLB

- Registers related to ITLB miss exceptions



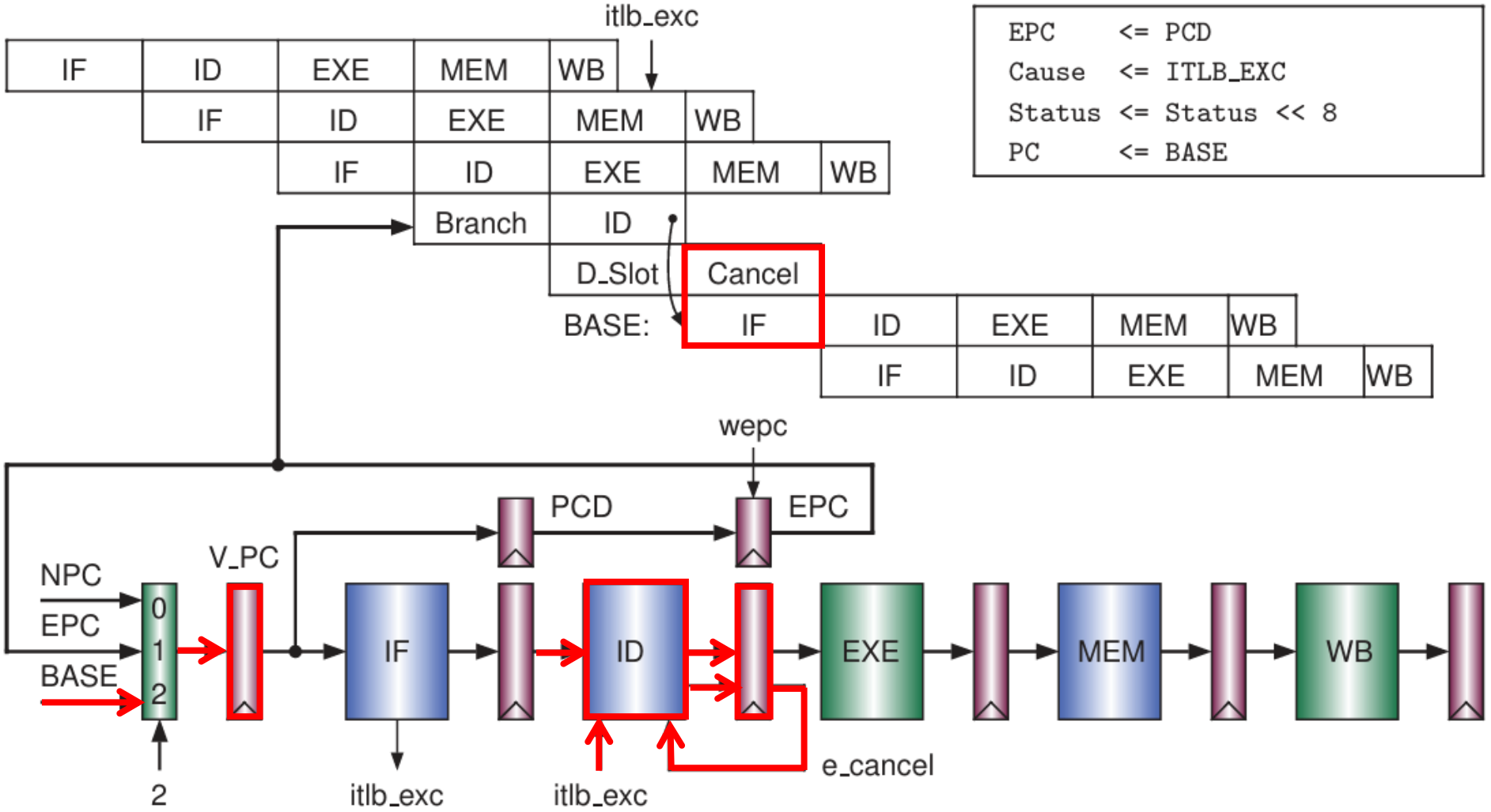
Design of Pipelined CPU with Caches and TLB

- Registers related to ITLB miss exceptions



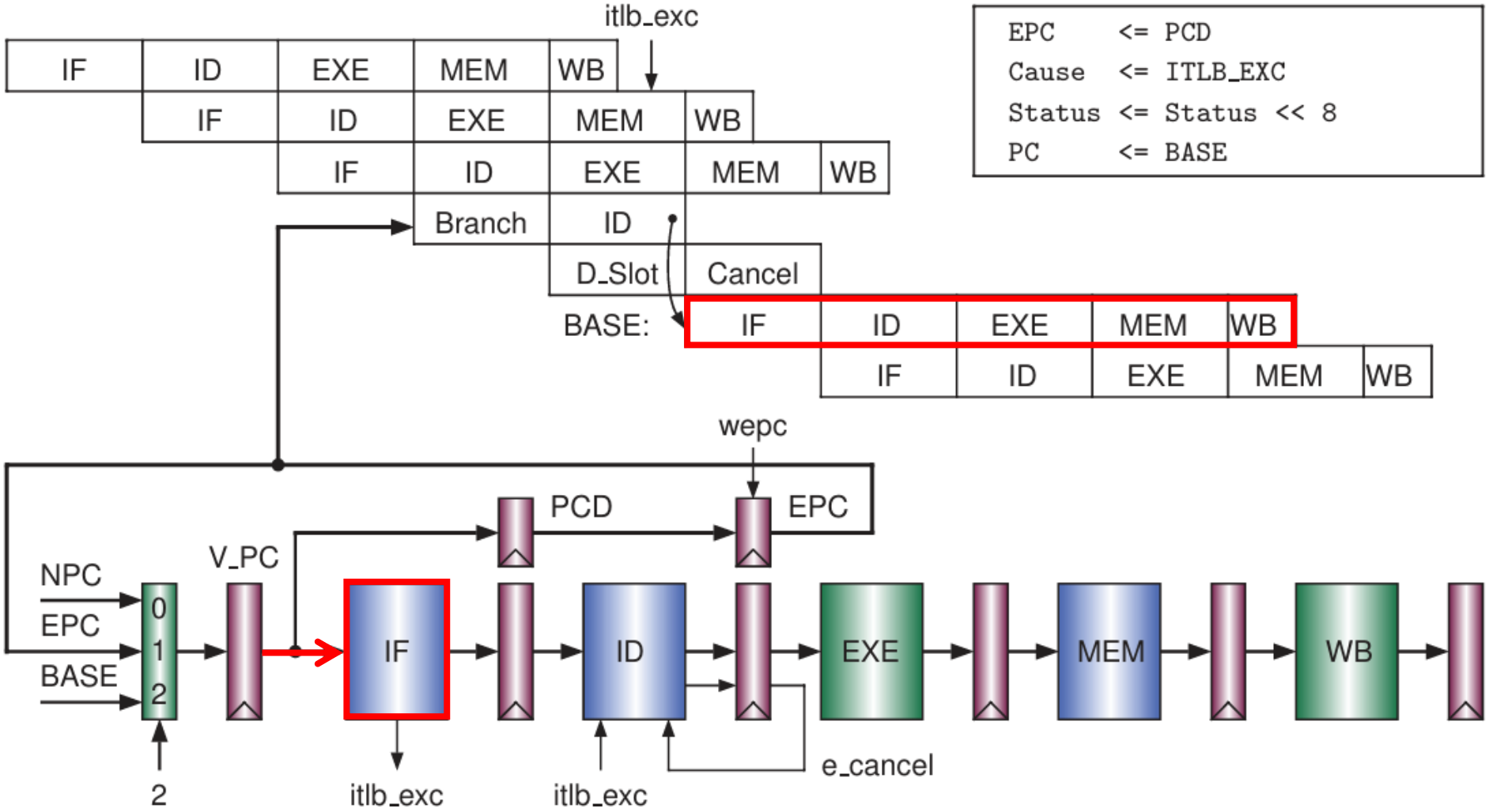
Design of Pipelined CPU with Caches and TLB

- Registers related to ITLB miss exceptions



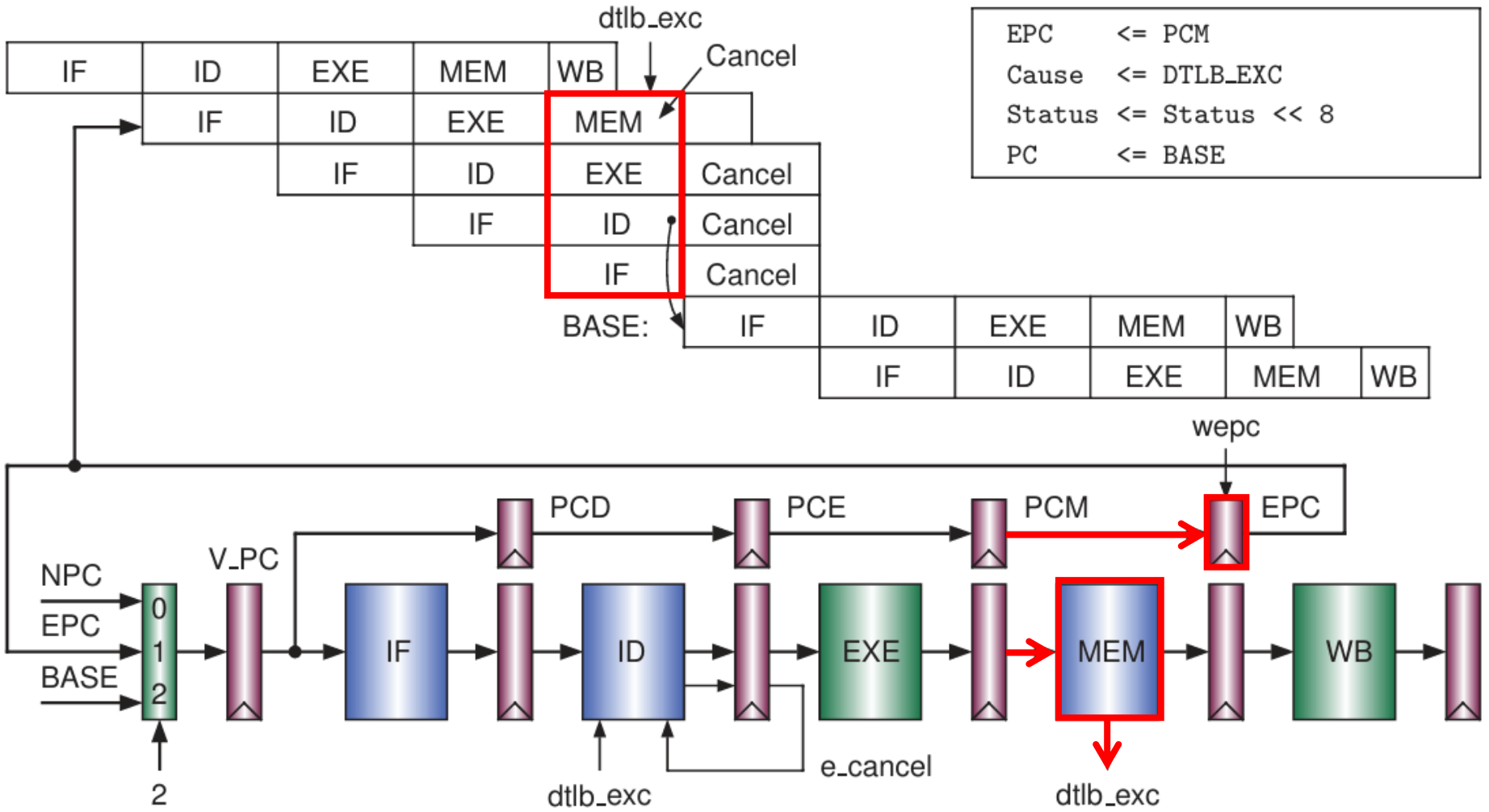
Design of Pipelined CPU with Caches and TLB

- Registers related to ITLB miss exceptions



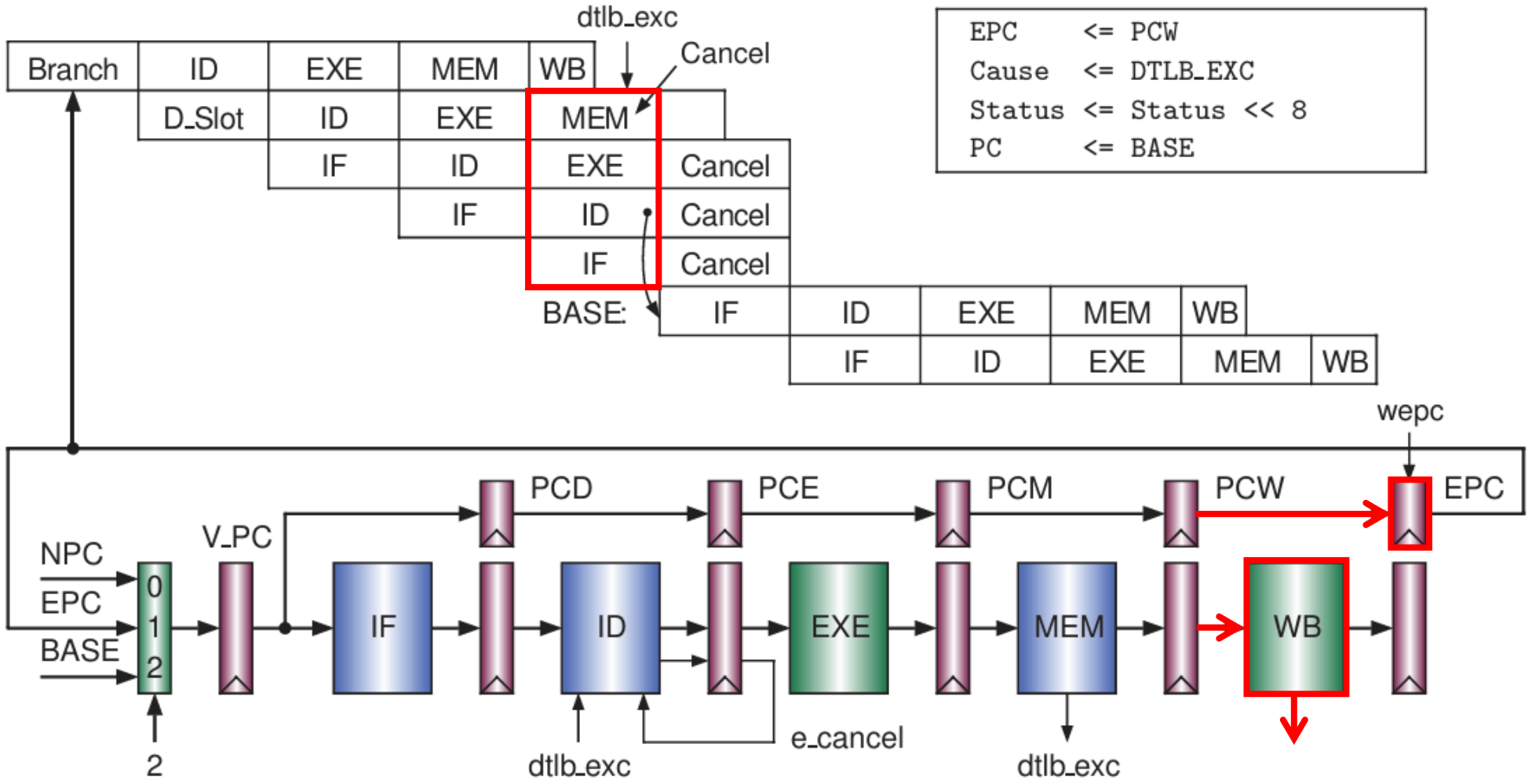
Design of Pipelined CPU with Caches and TLB

- Registers related to DTLB miss exceptions



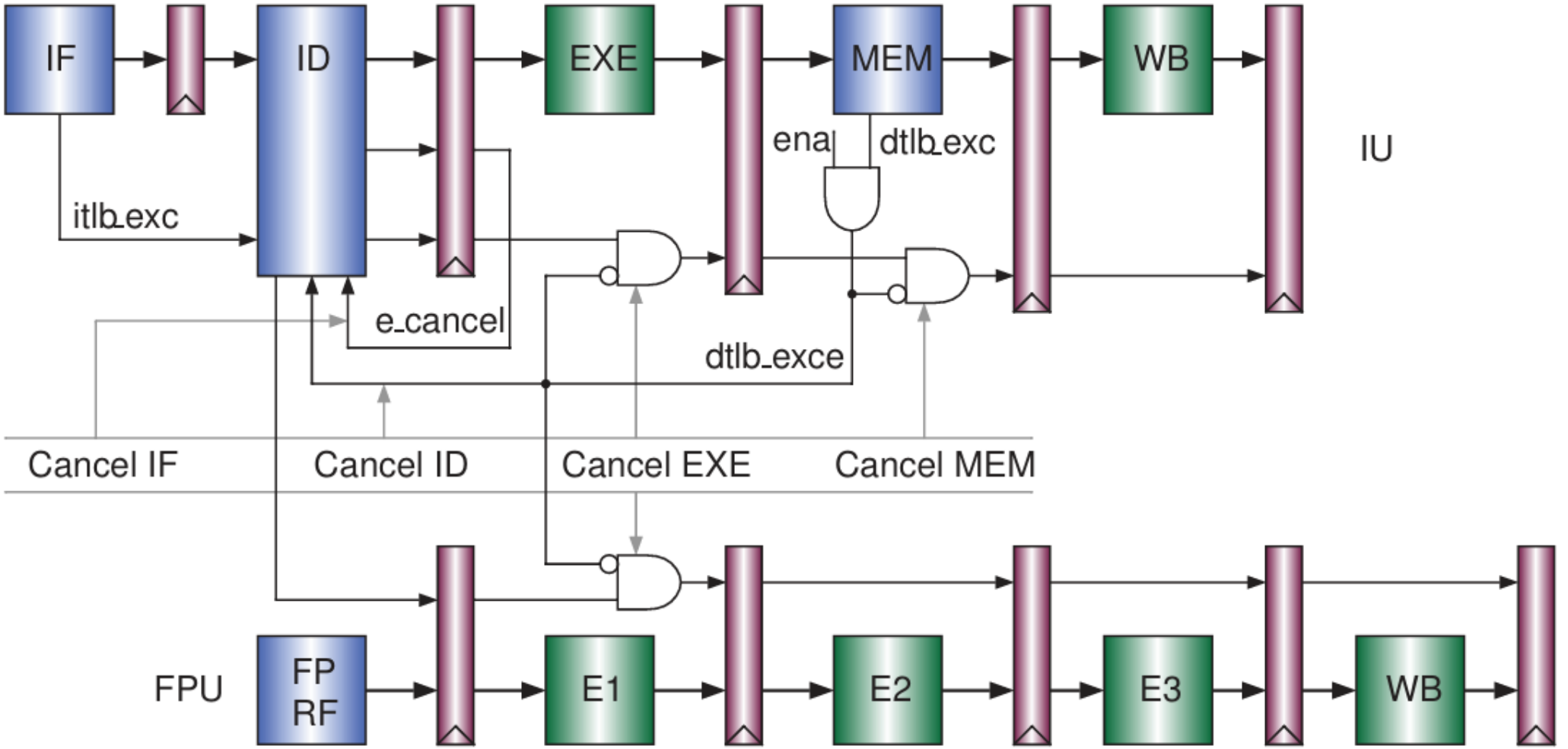
Design of Pipelined CPU with Caches and TLB

- Registers related to DTLB miss exceptions



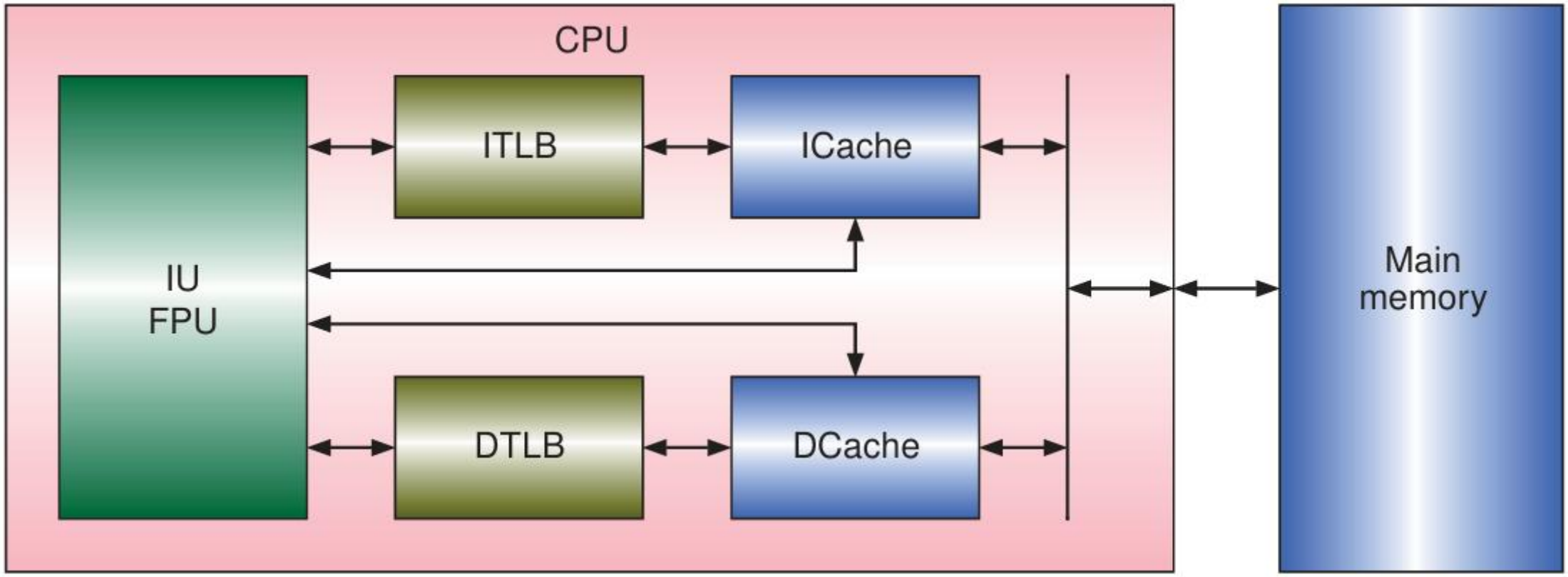
Design of Pipelined CPU with Caches and TLB

- Cancel instructions due to DTLB miss



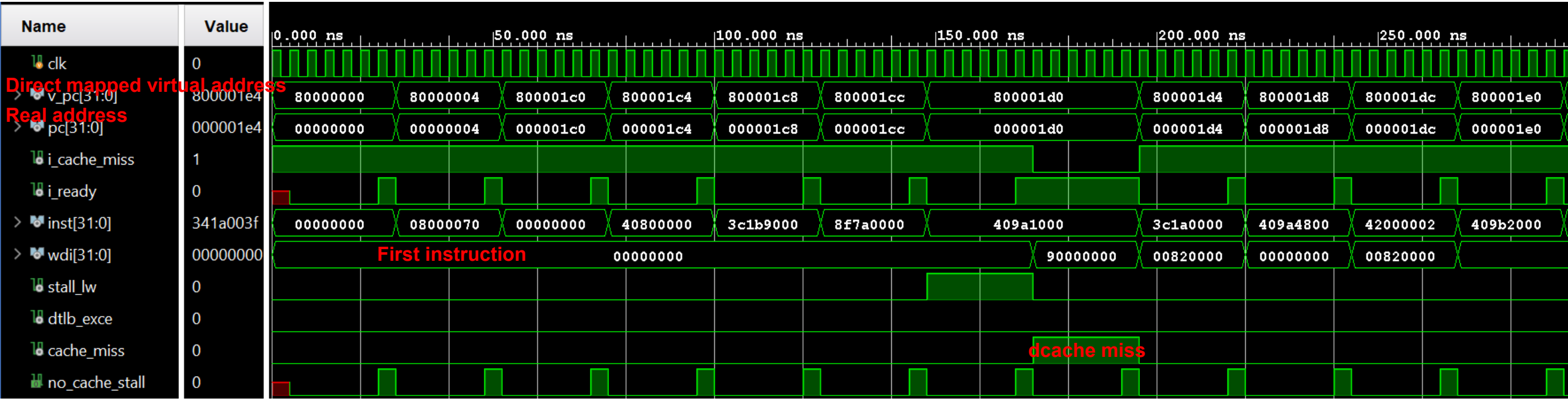
Design of Pipelined CPU with Caches and TLB

- Structure of CPU with caches and TLBs



Simulation

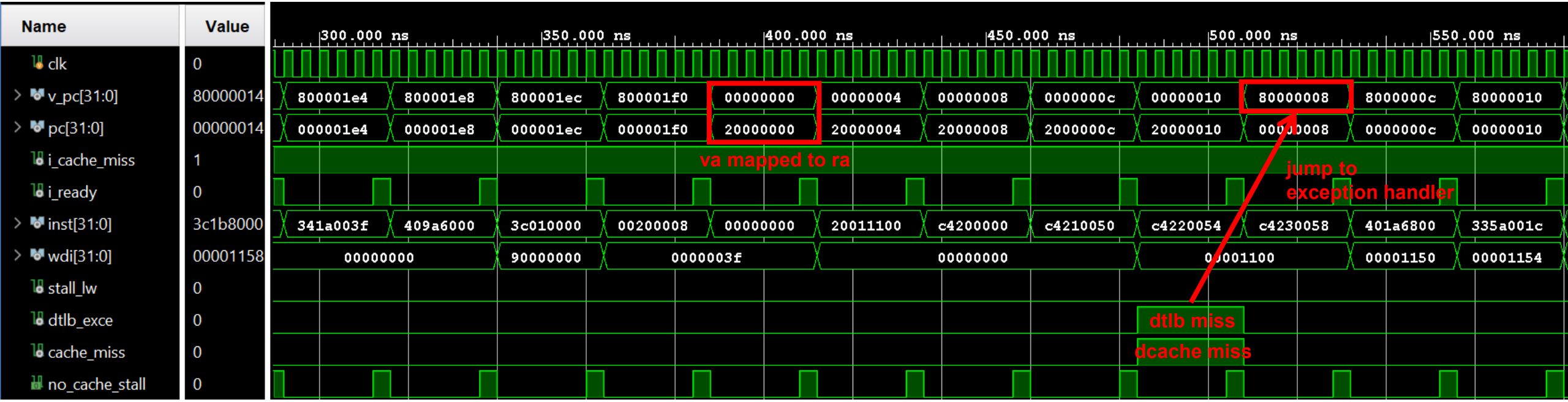
- Waveform of initializing ITLB




```
80000000 : 08000070; %  j    initialize_itlb  # jump to init itlb          %
80000004 : 00000000; %  nop                                                %
800001c0 : 40800000; %  mtc0 $0,  c0_index    # c0_index <-- 0 (itlb[0])   %
800001c4 : 3c1b9000; %  lui  $27, 0x9000      # page table base            %
800001c8 : 8f7a0000; %  lw   $26, 0x0($27)    # 1st entry of page table    %
800001cc : 409a1000; %  mtc0 $26, c0_entry_lo # c0_entrylo <-- v,d,c,pfn   %
800001d0 : 3c1a0000; %  lui  $26, 0x0         # va (=0) for c0_entry_hi    %
800001d4 : 409a4800; %  mtc0 $26, c0_entry_hi # c0_entry_hi <-- vpn (0)    %
800001d8 : 42000002; %  tlbwi                 # write itlb for user prog   %
800001dc : 409b2000; %  mtc0 $27, c0_context  # c0_context <-- ptebase     %
800001e0 : 341a003f; %  ori  $26, $0, 0x3f    # enable exceptions          %
```

Simulation

- Waveform of dealing with DTLB miss exception

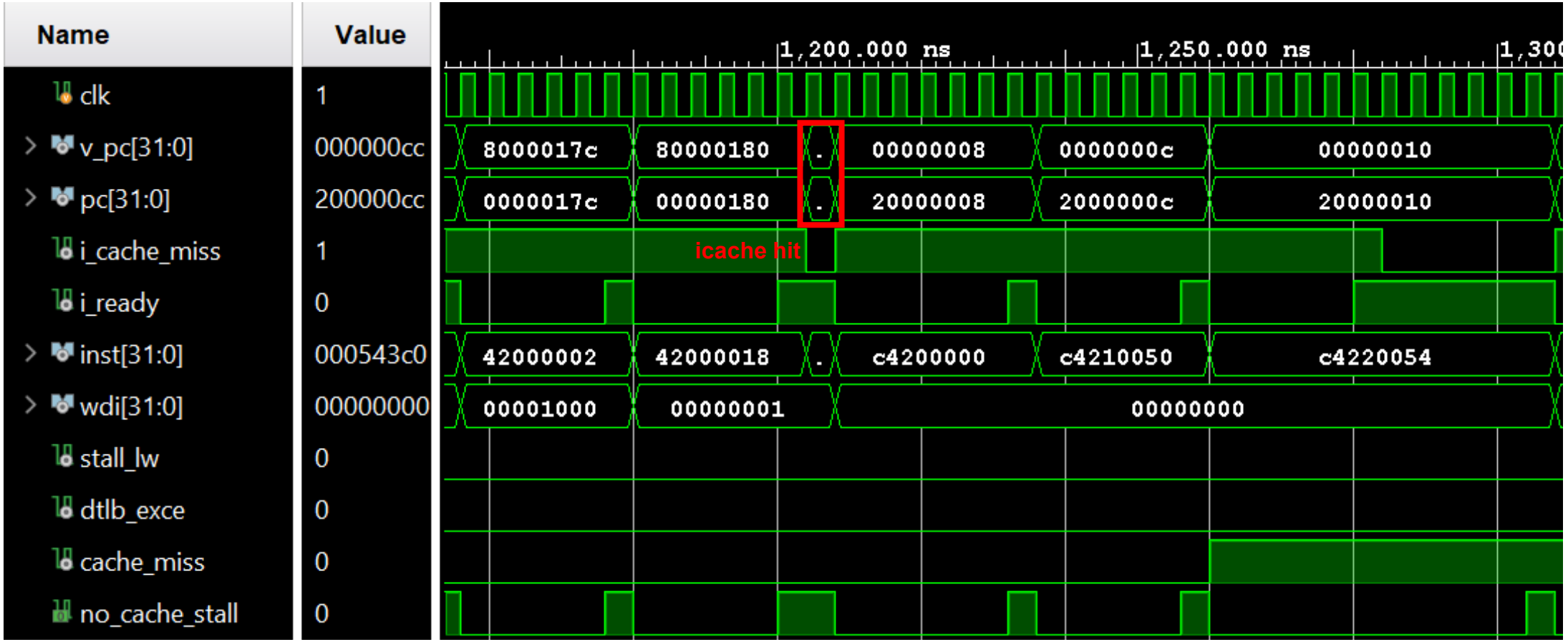



```
800001e4 : 409a6000; %  mtc0  $26, c0_status   # c0_status <-- 0..0111111  %
800001e8 : 3c010000; %  lui   $1,  0x0         # va = 0x0000_0000          %
800001ec : 00200008; %  jr    $1              # jump to user program      %
800001f0 : 00000000; %  nop                   #                           %
```

```
00000000 : 20011100; %  addi  $1,  $0,0x1100   # address of data[0]        %
00000004 : c4200000; %  lwc1  f0,  0x0($1)     # load fp data              %
00000008 : c4210050; %  lwc1  f1,  0x50($1)    # load fp data              %
0000000c : c4220054; %  lwc1  f2,  0x54($1)    # load fp data              %
00000010 : c4230058; %  lwc1  f3,  0x58($1)    # load fp data              %
80000008 : 401a6800; %  mfc0  $26, c0_cause    # read cp0 cause reg        %
8000000c : 335a001c; %  andi  $26, $26, 0x1c   # get exccode, 3 bits       %
```

Simulation

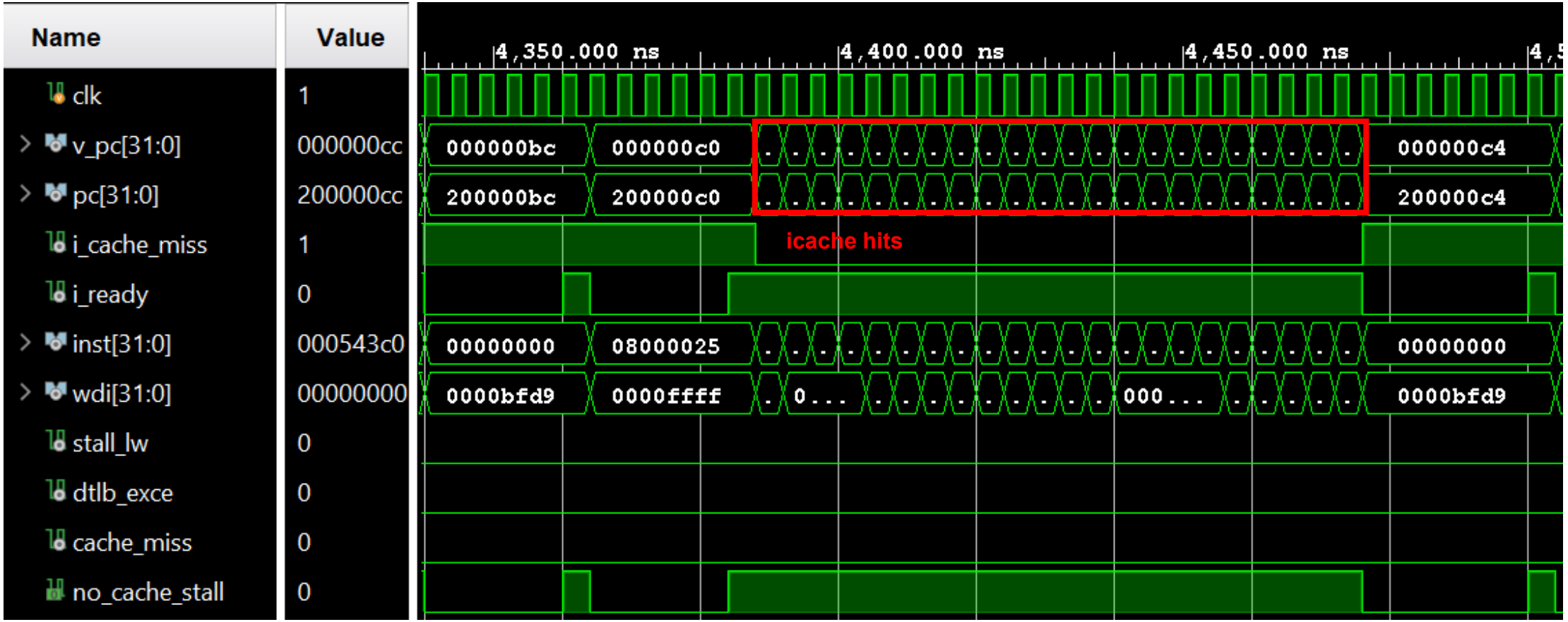
- Waveforms of returning from exception and cache hits




```
8000017c : 42000018; %  eret                 # return from exception
80000180 : 00000000; %  nop                  #
00000004 : c4200000; %  lwc1  f0,  0x0($1)   # load fp data
% 00000008 : c4210050; %  lwc1  f1, 0x50($1)   # load fp data
% 0000000c : c4220054; %  lwc1  f2, 0x54($1)   # load fp data
% 00000010 : c4230058; %  lwc1  f3, 0x58($1)   # load fp data
```

Simulation

- Waveforms of returning from exception and cache hits



References

[1] Yamin Li, *Computer Principles and Design in Verilog HDL*, Wiley, 2016.