Motorola MPC7451 Processor

Overview

- PowerPC history
- Microarchitecture
- Memory management
- Programming model
- Multimedia extensions

PowerPc History

- **PowerPC** is a RISC microprocessor architecture created by the 1991 Apple-IBM-Motorola alliance
- Originally intended for personal computers, have become popular for embedded and high-performance processors
- PowerPC is largely based on IBM's earlier POWER (***P**erformance **O**ptimization **W**ith **E**nhanced **R**ISC*) architecture
- The PowerPC is designed along RISC principles, and allows for a superscalar implementation.
- Versions of the design exist in both 32-bit and 64-bit implementations

PowerPc History (2)

- 601 (1992) MPC601 50 and 66 MHz
- 602 consumer products (multiplexed data)
- 603 notebooks
- 604
- 620 the first 64-bit implementation
- 750 PowerPC G3 (1997) 233 MHz e 266 MHz
- 7400 PowerPC G4 (1999) 350 MHz
- 750FX announced by IBM in 2001 and available early 2002 at 1 GHz .
- 970 PowerPC G5 (2003) A 64-bit implementation derived from the IBM POWER4 operating at speeds of 1.4 GHz, 1.6 GHz, 1.8 GHz, 1.9 GHz, 2.0 GHz, 2.1 GHz, 2.3 GHz, 2.5 GHz and 2.7 GHz
- 970MP G5 Dual Core Processor available late 2005 at speeds of da 2.0, 2.3 e 2.5 GHz.

PowerPc Future

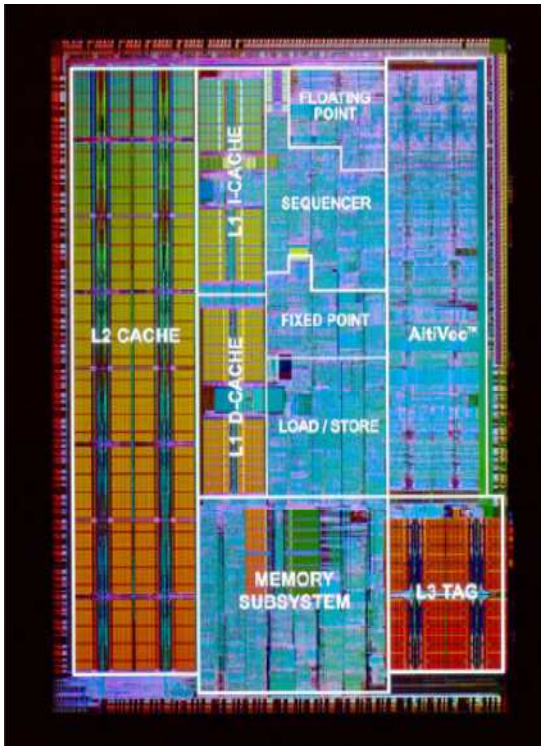
- On June 7, 2005, during the World Wide Developers Conference 2005 (WWDC2005), Steve Jobs announced that Apple will be dropping the PowerPC line of processors.
- The first Intel Macs were released on January 10, 2006 and Apple has indicated that they will continue to replace current PowerPC-based models with Intel-based models.
- IBM focused their chip designs for PowerPC CPUs towards game machine makers such as Nintendo's Wii, Sony's PlayStation 3 and Microsoft's Xbox 360.

MPC7451: Functional Features

- 32-bit implementation of the PowerPC RISC architecture.
- High-frequency superscalar G4 core.
- Seven-stage pipeline with 11 execution units.
- On chip cache L1 and L2, support for L3 cache.
- 128-bit implementation of Motorola’s AltiVec™.

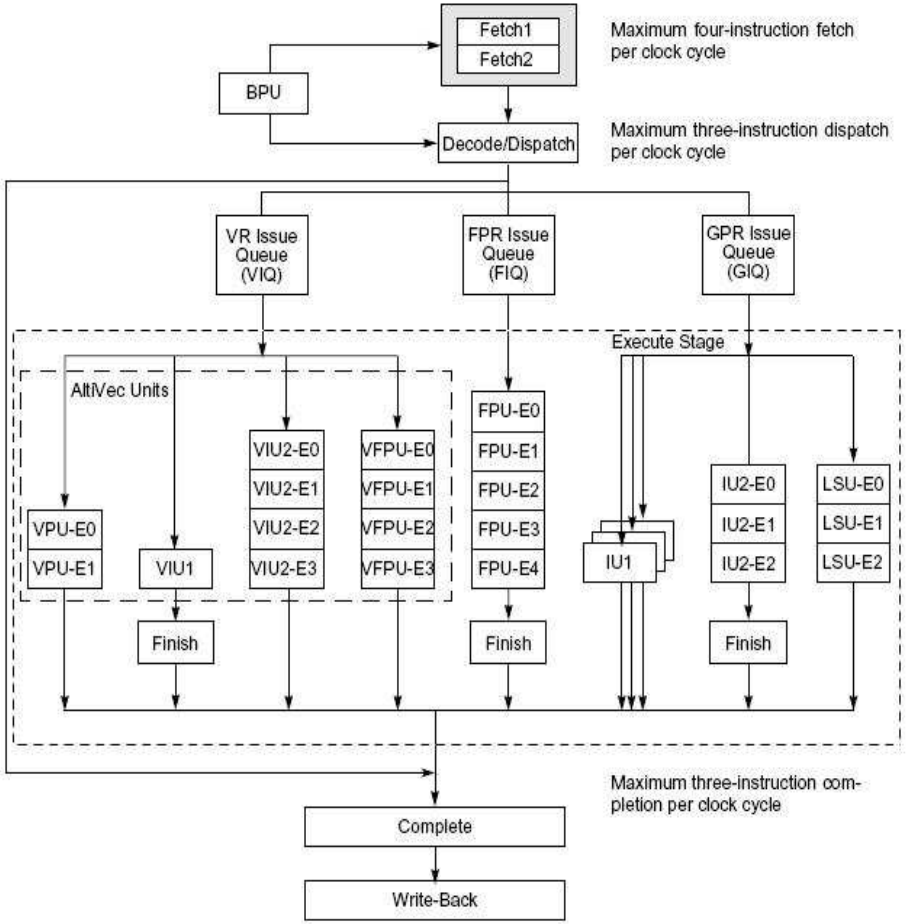
MPC7451: Technological Features

- CPU Speeds 533, 667, 733, and 867 MHz
- Synchronous Bus at frequency of 133 MHz, Bus Interface 64-bit (MPX/60x).
- Transistor count 33 million
- Power Dissipation 17W @ 533 MHz
- Die Size 106 mm2, Package 483 CBGA
- Process 0.18µ CMOS
- 2.1 MIPS @ 733 MHz



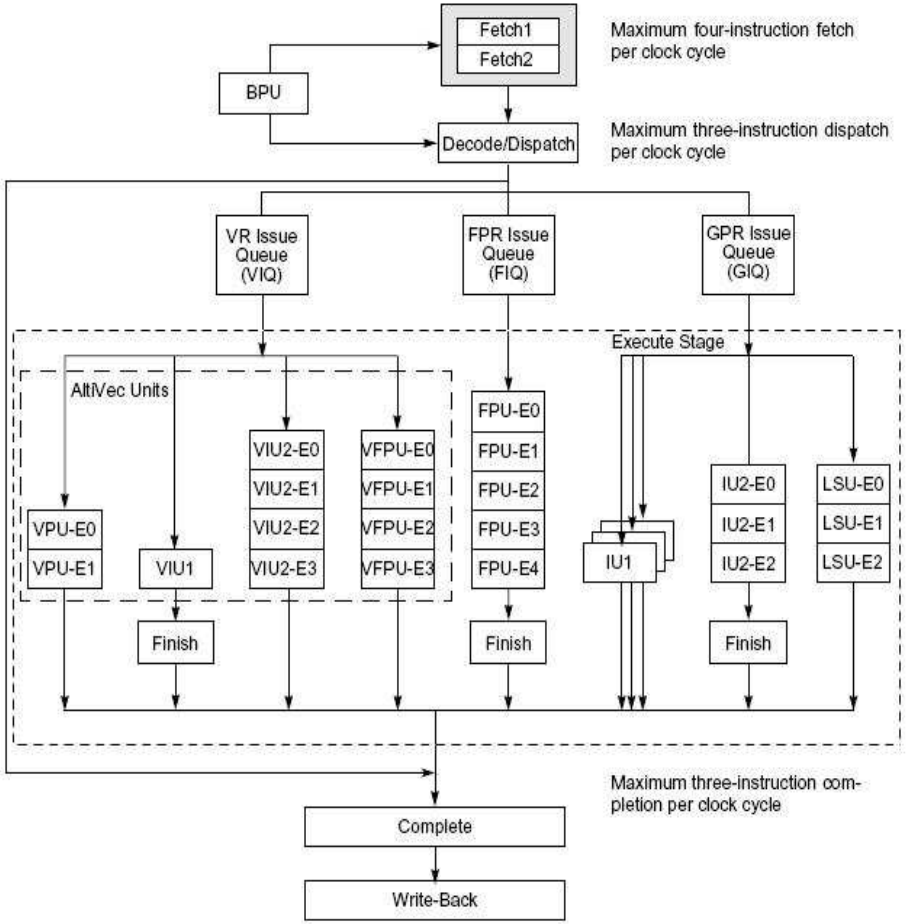
Seven-stage Master Pipeline

- Several execution units feature multiple-stage pipelines.
- Typically, instructions follow one another through the stages, so a four-stage unit can work on four instructions when its pipeline is full.
- The execution unit can achieve a throughput of one instruction per clock cycle.

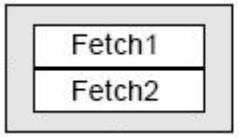


Seven-stage Master Pipeline

- Master pipeline:
  - Fetch1
  - Fetch2
  - Decode/Dispatch
  - Issue
  - Execute
  - Complete
  - Write-Back



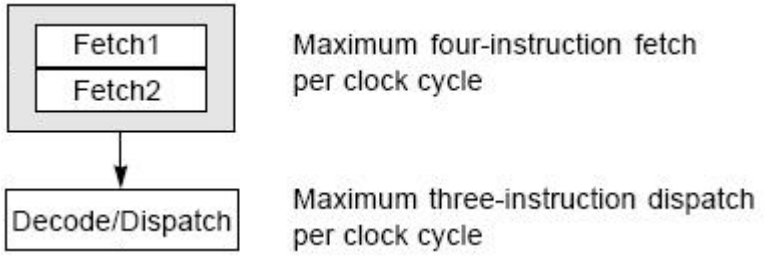
FETCH



Maximum four-instruction fetch per clock cycle

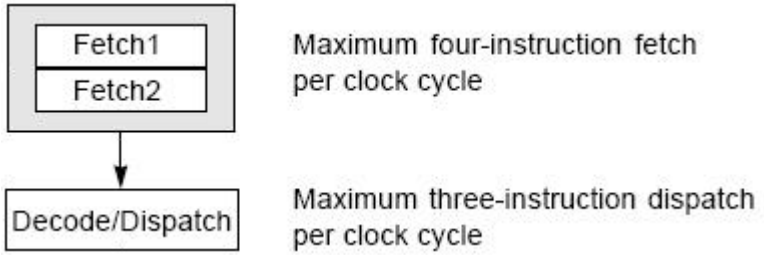
- An instruction fetch consists of two single-cycle fetch stages.
- Instructions are fetched from memory and placed in the 12-entry IQ (Instruction Queue).
- As many as four instructions can be fetched into the IQ during each clock cycle.
- Fetcher tries to initiate a fetch every cycle only if IQ8–IQ11 are empty.
- Latency may vary: in the best case the instructions arrive two clock cycles later (cache hit and cache idle).

DECODE/ DISPACH



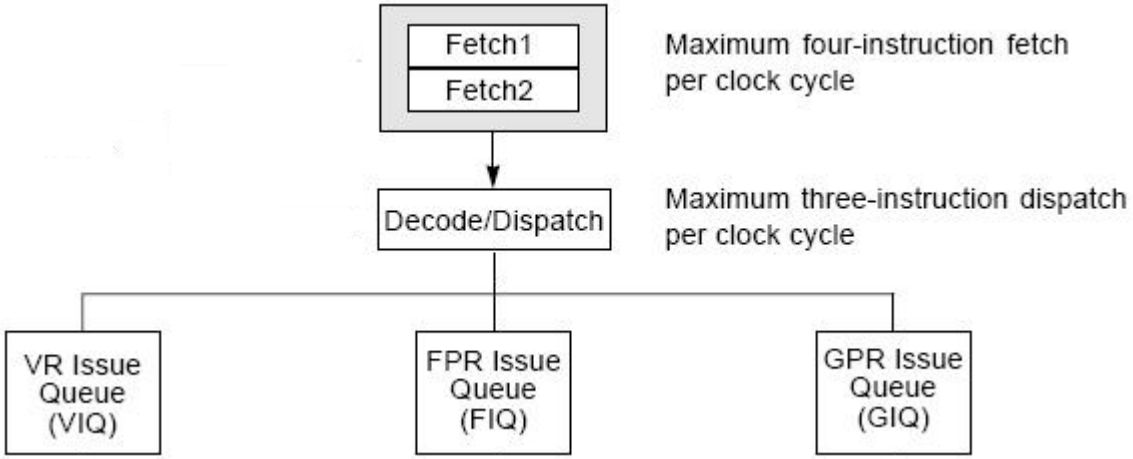
- The decode/dispatch stage fully decodes each instruction; the instruction passes to the appropriate issue pipeline stage by taking a place in the completion queue
- The dispatch unit:
  - checks for source and destination register dependencies.
  - determines whether a position is available in the CQ.
  - Inhibits subsequent instruction dispatching as required.
- As many as three instructions can be dispatched per clock cycle (IQ0–IQ2), but they **cannot** be dispatched out of order.

DECODE/ DISPACH (2)



- Can send three instructions to the various issues queues (maximum three to the GIQ, two to the VIQ, one to the FIQ and one load/store instruction per cycle).
- As many as four GPRs, three VRs, and two FPRs can be renamed per cycle.
- Resources required to avoid stalls in the dispatch unit:
  - Appropriate issue queue is available.
  - The CQ is not full.
  - Previous instructions the IQ dispatch entries must dispatch.
  - Needed rename registers are available.

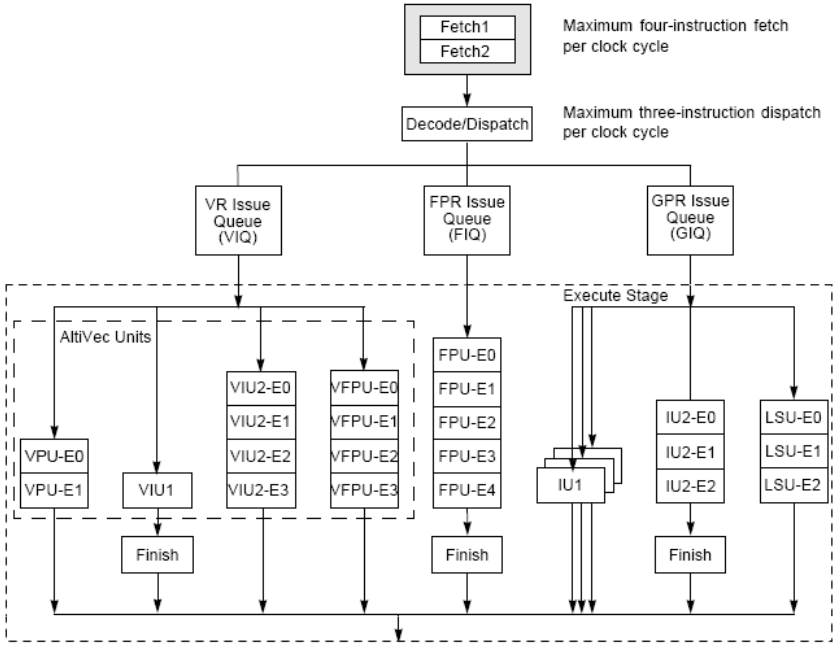
ISSUE



- The issue stage reads source operands from rename registers and register files and determines when instructions are latched into the execution unit reservation stations.
  - As many as three instructions can be issued **out-of-order** to the LSU, IU2, and IU1 reservation stations from the bottom three GIQ entries.
  - As many as two instructions can be issued **in-order** to the VPU, VIU2, VIU1 and VFPU reservation stations from the bottom two VIQ entries.
  - As many as one instruction can be issued to the FPU reservation station from the FIQ entry.

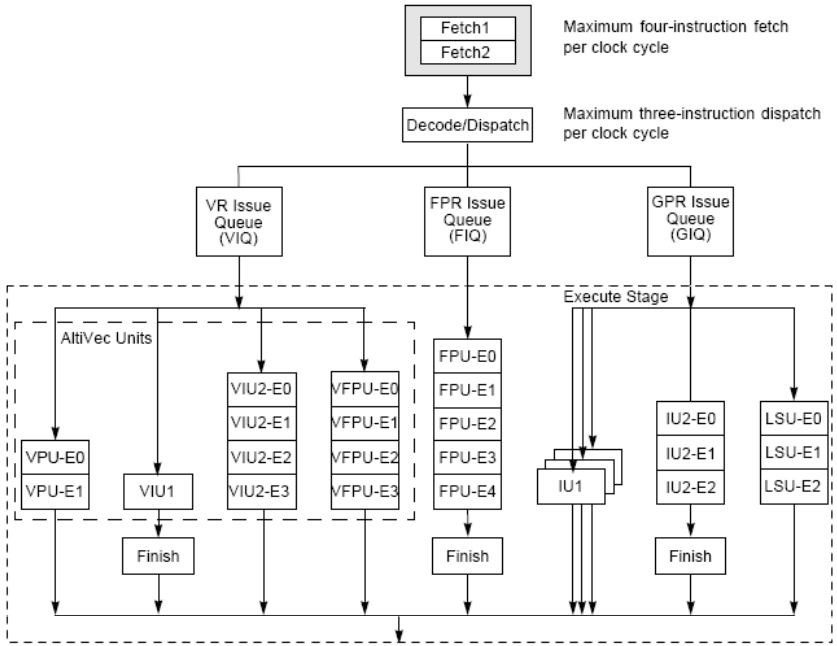
- GIQ: 6-Entry/3-Issue General Purpose Register Issue Queue.
- VIQ: 4-Entry/2-Issue Vector Register Issue Queue, for handling AltiVec instructions.
- FIQ: 2-Entry/1-Issue Floating Point Register Issue Queue, for floating point instructions.

EXECUTE



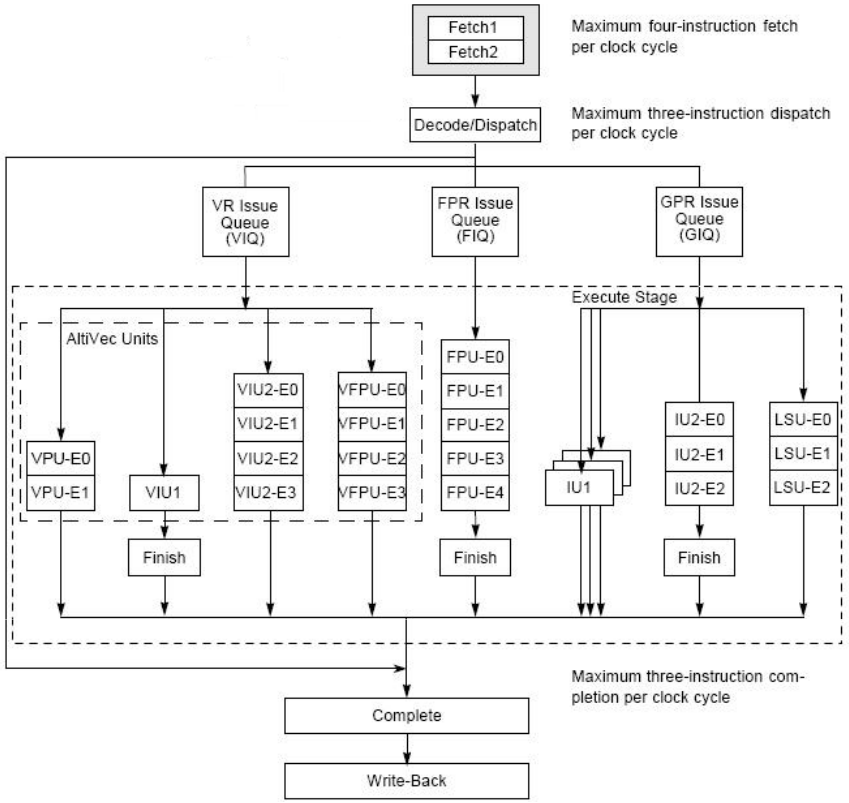
- The execution unit executes the instruction, writes results on its result bus, and notifies the CQ (Completition Queue) when the instruction finishes.
  - Three IU1s execute one cycle integer instructions (all except multiply, divide, and move to/from SPR).
  - IU2 executes miscellaneous integer instructions with latencies of 3 cycles or more.

EXECUTE (2)



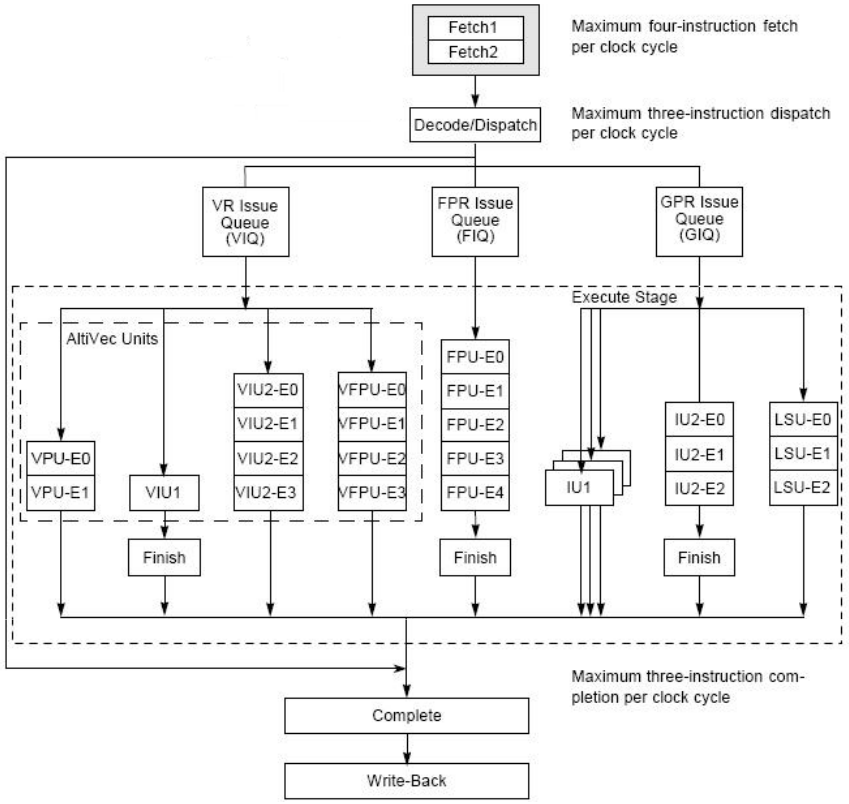
- FPU executes single-and double-precision operations with a latency of five cycles.
- LSU executes all load and store instructions with a latency of three cycles or more.
- AltiVec Unit: (VPU, VIU1, VIU2, VFPU).

COMPLETE / WRITE-BACK



- Complete:
  - Retires as many as three instructions per cycle from CQ in program order.
  - Requirements for retiring instructions:
    - Instructions must have finished execution.
    - Previous instructions must be retired.
    - Instructions must not follow an unresolved predicted branch.

COMPLETE / WRITE-BACK (2)

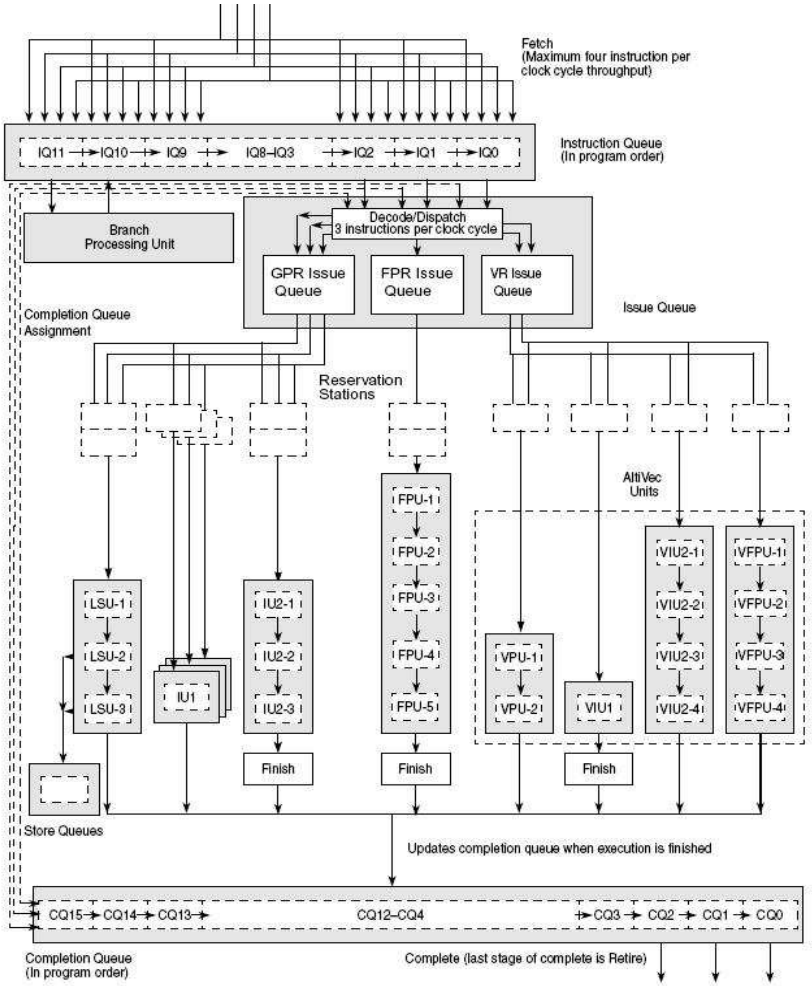


- Write-Back:
  - Commits execution results to architected registers (GPRs, FPRs,VRs, LR, and CTR).
  - Only three renames of a given type can be retired per cycle.

- Ensure in-order completion and a precise model exception:

  → guarantee correct architectural state when recover from a mispredicted branch or exception.

Istruction Flow Diagram



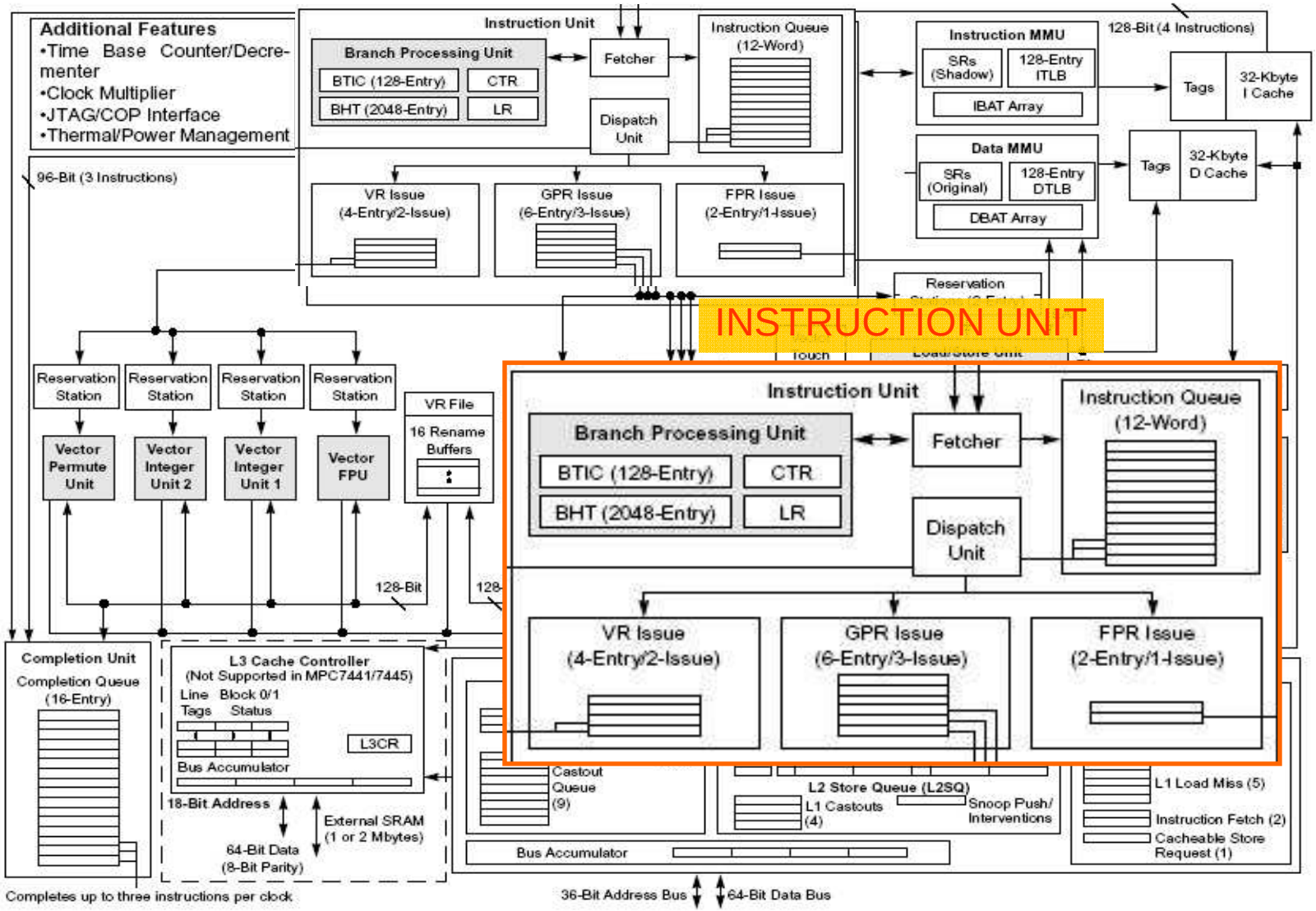
• <u>Rename registers:</u>
hold instruction results before the completion commits them to the architecture defined register

⇨ avoid WAR and WAW hazard.

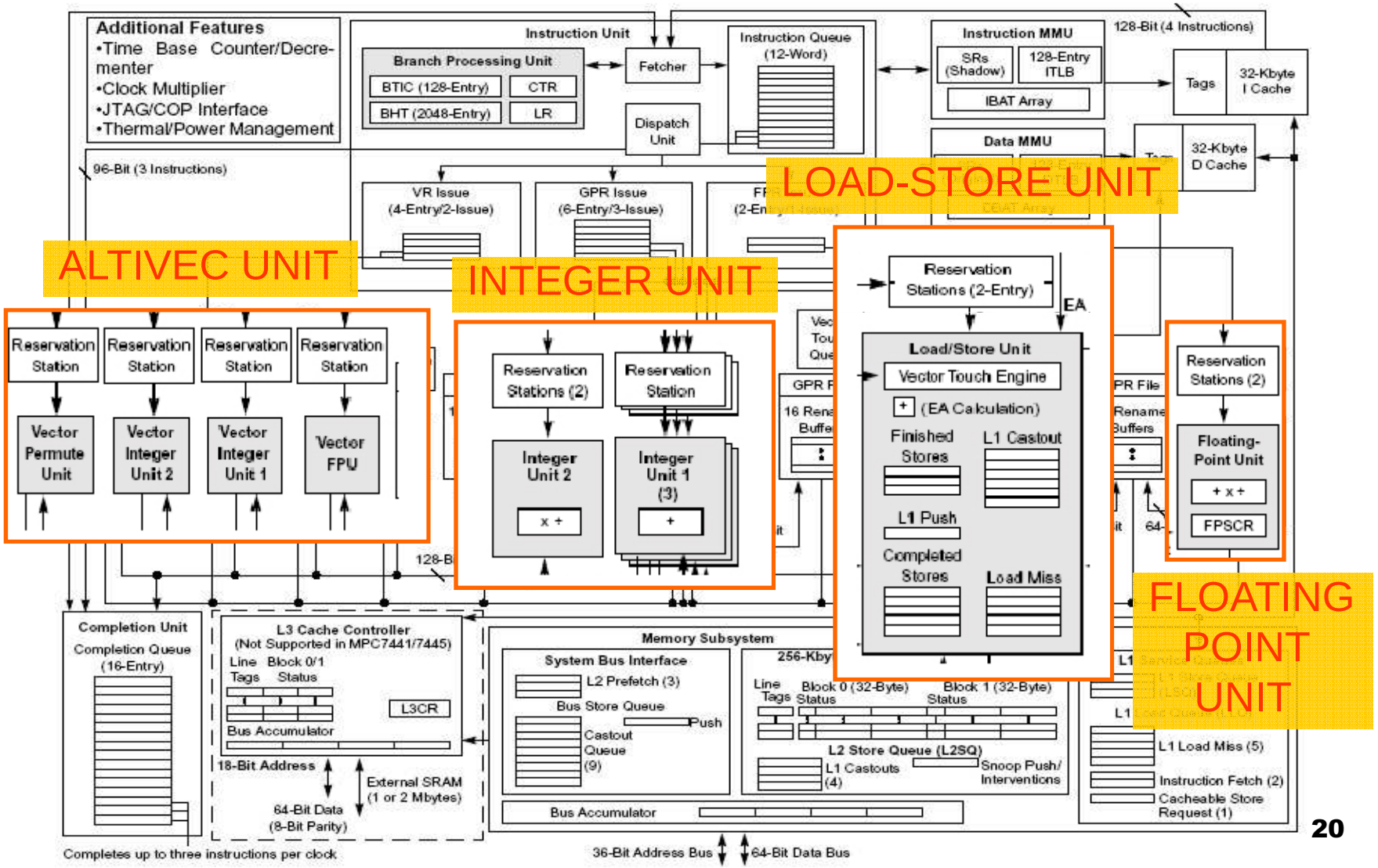
• <u>Reservation station:</u>
a buffer (in front of one or more FUs) with one or more entries and each entry can buffer an instruction with its operands

⇨ avoid data dependencies.

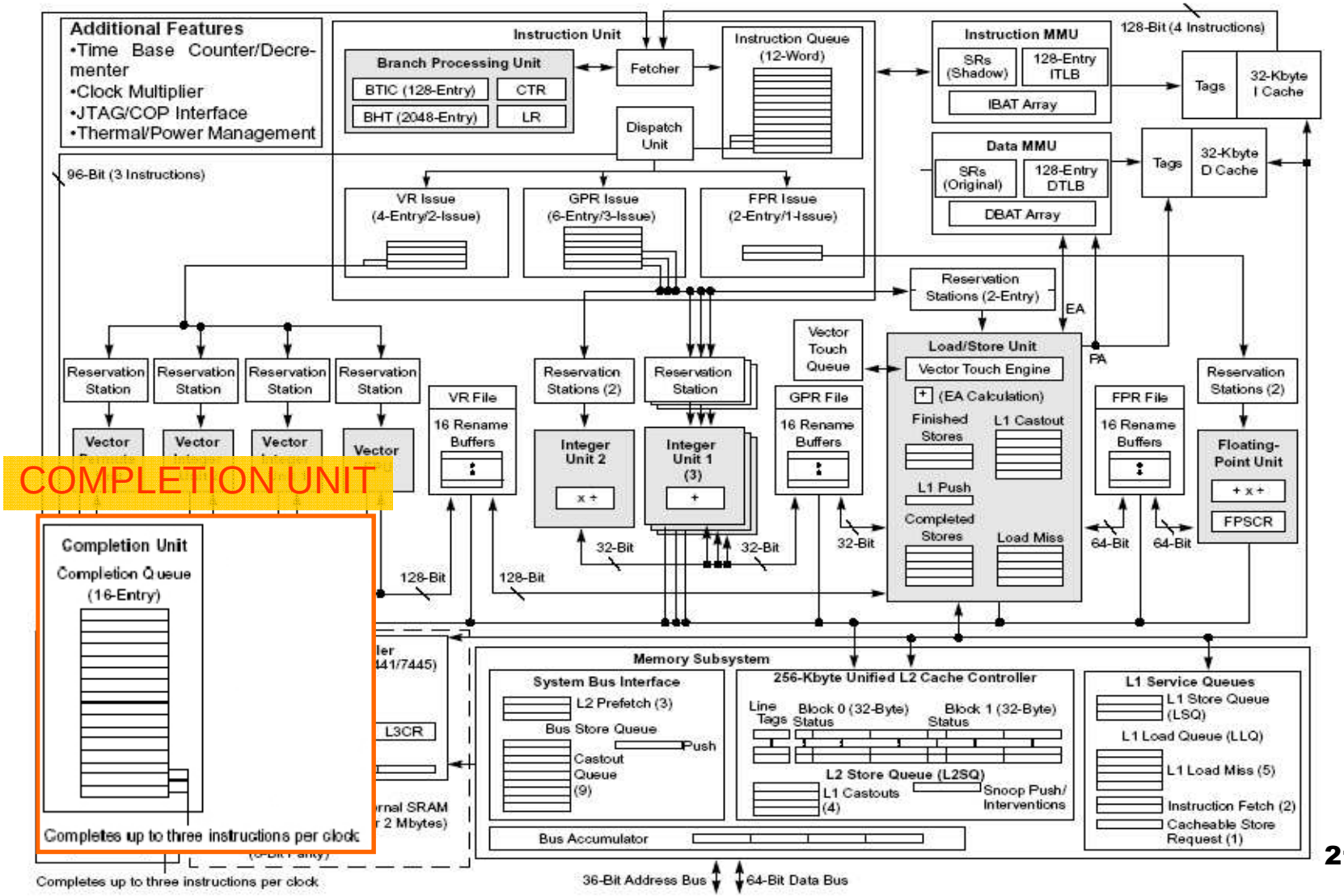
MPC7451 Block Diagram



MPC7451 Block Diagram



MPC7451 Block Diagram



Branch Prediction Unit

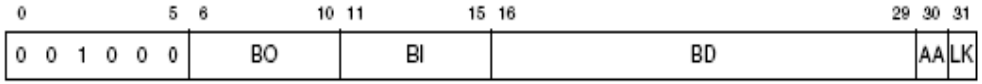
- The BPU receives branch instructions from the IQ (IQ0-IQ7) and executes them.
- Branches with no outstanding dependencies (CR, LR, CTR) are resolved immediately.
- Except those that update the LR or CTR, most branch instructions are removed from the instruction flow before they take a position in the CQ.
- For branches in which the direction is unresolved the branch path is predicted using:
  - Static branch prediction
  - Dynamic branch prediction.
  - Btic
  - Link stack prediction

Branch Prediction Unit (2)

- Unresolved branches are held in a three-entry branch queue ⇒ The MPC7451 executes through three prediction levels.
- When a prediction is made, instruction fetching, dispatching, and execution continue from the predicted path, but instructions **cannot** complete and write back results to architected registers until the prediction is determined to be correct (resolved).
- When a prediction is incorrect, the instructions from the incorrect path are flushed from the processor and processing begins from the correct path.

STATIC BRANCH PREDICTION

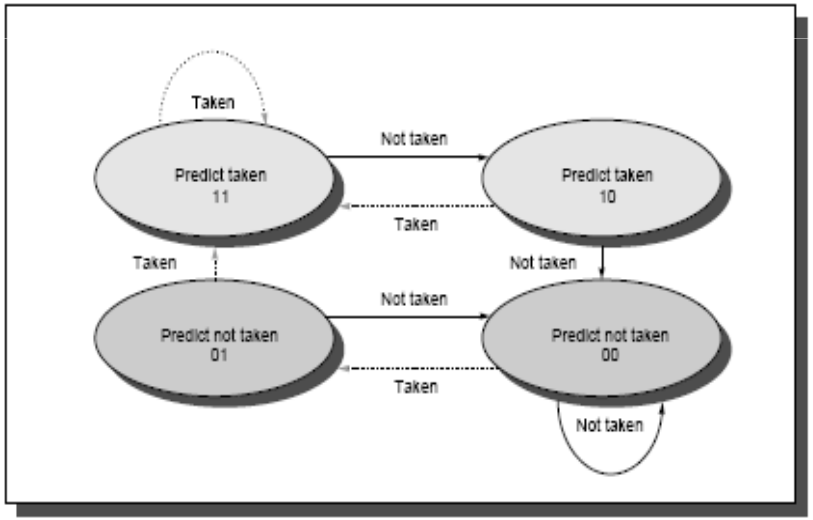
- The PowerPC architecture provides a field in branch instructions (the BO field) to allow software to hint whether a branch is likely to be taken.
  - The first four bits of the BO operand specify how the branch is affected by or affects the CR and CTR. The fifth bit (*y*), is used for branch prediction.
  - Es: bc istruction format



- Clearing the *y* bit indicates a predicted behavior for the branch instruction as follows:
  - For **bc***x* with a negative value in the displacement operand, the branch is taken.
  - In all other cases (**bc***x* with a non-negative value in the displacement operand, **bclr***x*, or **bcctr***x*), the branch is not taken.
- Setting the *y* bit reverses the preceding indications.
- The default value for the *y* bit should be 0.

DYNAMIC BRANCH PREDICTION

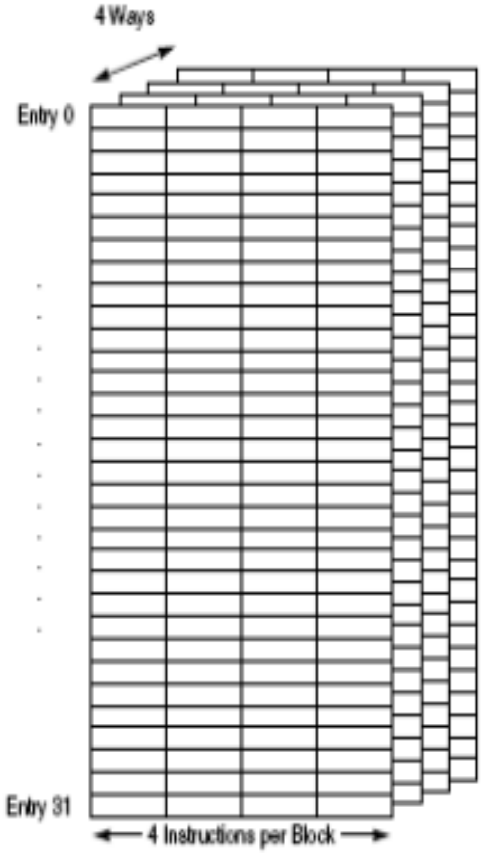
- Dynamic prediction is implemented using a 2048-entry branch history table (BHT), a cache that provides two bits per entry that together indicate four levels of prediction for a branch
  - not-taken,
  - strongly not-taken,
  - taken,
  - strongly taken



- To reduce aliasing, only predicted branches update BHT entries.

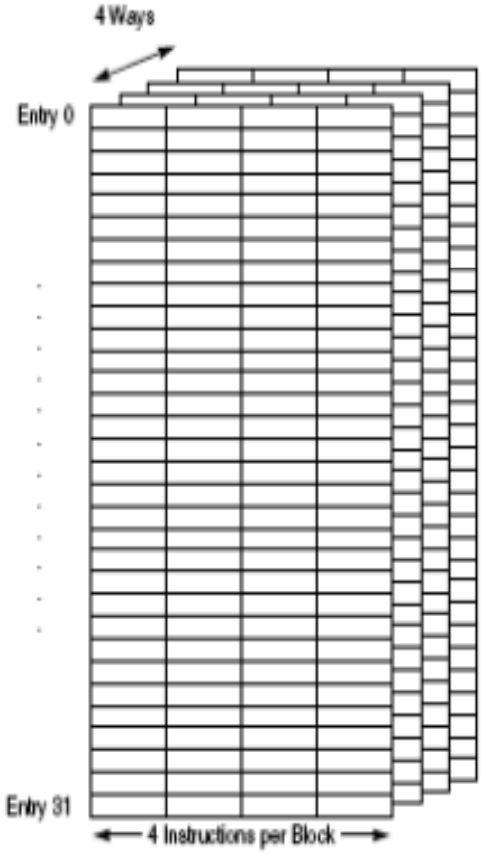
BTIC

- The BTIC is a 128-entry, four-way set associative cache that contains the most recently used branch target instructions (up to four instructions per entry) for **b** and **bc** branches.
- When a taken branch instruction hits in the BTIC, the instructions arrive in the IQ two clock cycles later, a clock cycle sooner than they would arrive from the instruction cache.



BTIC (2)

- BTIC entries are indexed not from the address of the first target instruction but from the address of the branching instruction, so multiple branches sharing a target generate duplicate BTIC entries.
- BTIC is virtually addressed. Because the BTIC is automatically flushed any time the address mappings might change, aliases do not occur in the BTIC.
- BTIC ways are updated using a FIFO algorithm.



LINK STACK PREDICTION

- The MPC7451 also avoids stalls by implementing an eight-entry branch link stack. As many as eight levels of **bclr**/branch-and-link pairs can be held and the **bclr** target address can be predicted from the link stack rather than requiring a stall until the **ld**/**mtlr** subroutine restore sequence completes.
- To use the link stack correctly, each branch-and-link instruction must be paired with a branch-to-link instruction.

MEMORY SUBSYSTEM

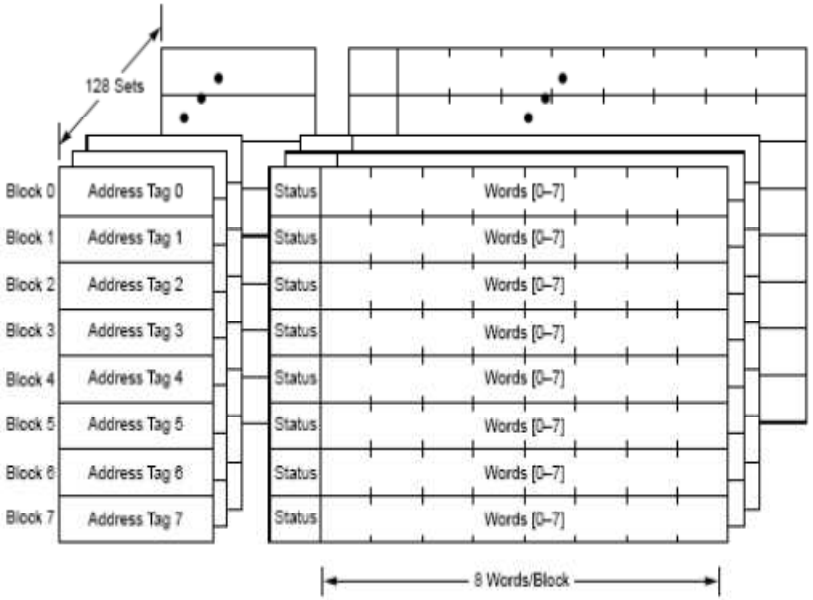
- Virtual memory support for up to 4 Petabytes (2^{52}) of virtual memory and real memory support for up to 64 Gigabytes (2^{36}) of physical memory.
- L1 instruction cache: 32-Kbyte and eight-way set associative
- L1 data cache: 32-Kbyte and eight-way set associative
- Unified L2 cache: 256-Kbyte and eight-way set associative
- Unified L3 cache interface: eight-way set associative tag memory to support up to 2-Mbyte external SRAM
- Separate MMUs for instruction and data

CACHE OVERVIEW

- Caches are physically addressed
- Cache write-back or write-through operation programmable on a per-page or per-block basis
- Instruction cache can provide four instructions per clock cycle
- Data cache can provide four words per clock cycle
- Two-cycle latency and single-cycle throughput for instruction or data cache accesses.
- Caches can be disabled/locked in software
- Supports a four-state modified/exclusive/shared/invalid (MESI) coherency protocol.

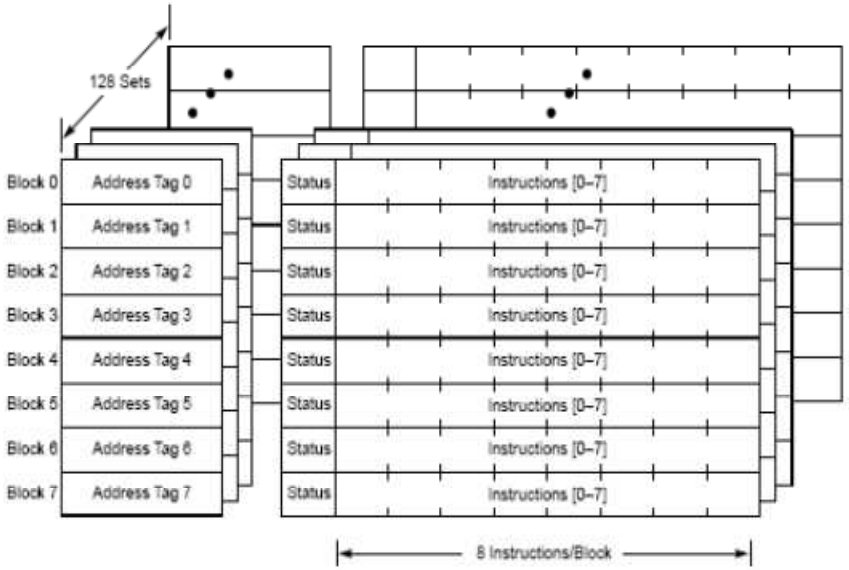
L1 DATA CACHE

- 128 set of eight blocks
- Each line consist of:
  - Address tag
  - Eight contiguous word (32-byte)
  - two status bit (MESI)
  - Four parity bits/word
- Non-blocking: allow hits under misses
- PLRU replacement algorithm



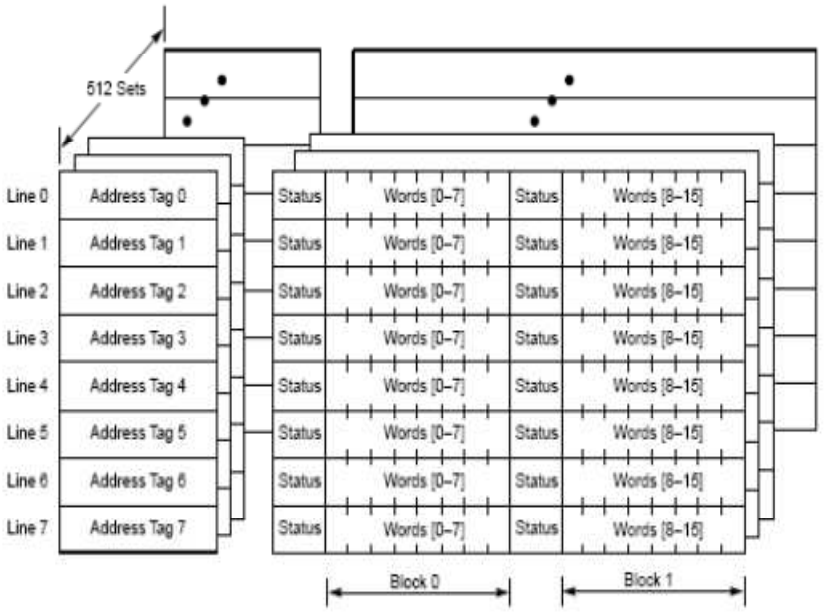
L1 INSTRUCTION CACHE

- 128 set of eight blocks
- Each line consist of:
  - Address tag
  - Eight contiguous instruction (32-byte)
  - No support for multiple state cache protocol
  - Single valid status bit
  - One parity bit/word
- Non-blocking: allow hits under misses, and misses under misses
- PLRU replacement algorithm



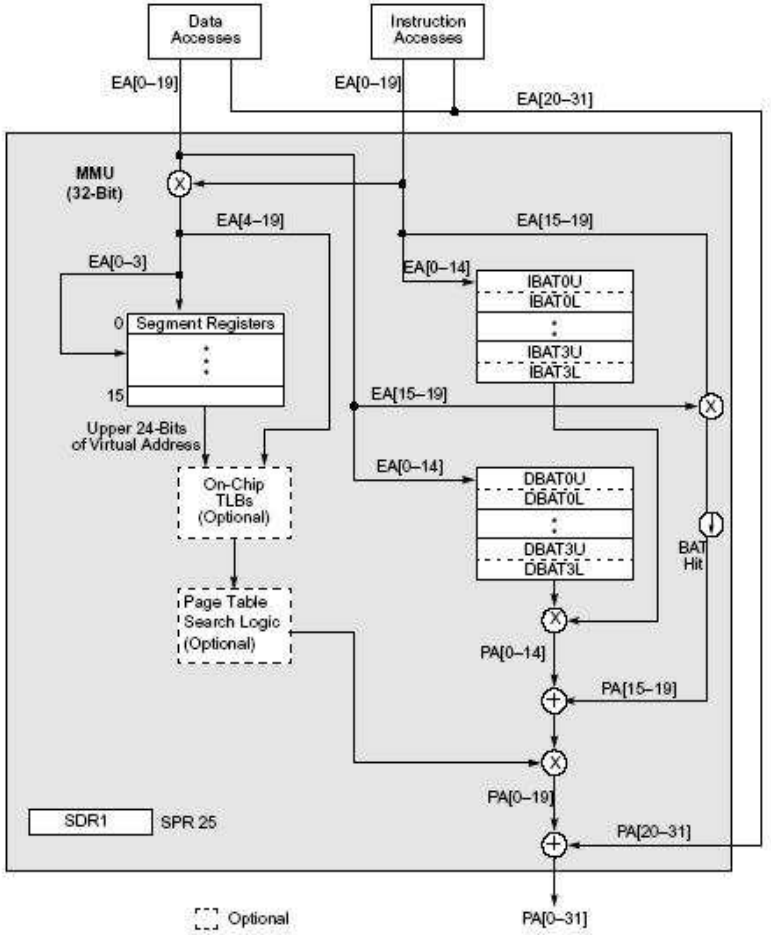
L2 CACHE

- 512 set of eight blocks
- Each line consist of:
  - Address tag
  - 16 contiguous words (64-bytes)
  - two separate status bits for the 8 words of the cache block
  - Support for MESI protocol
  - One parity bit/byte and an additional parity bit for each tag
- Non-blocking: allow hits under misses
- Supports two pseudo-random modes of line replacement:
  - 3-bit counter mode
  - pseudo-random number generator mode



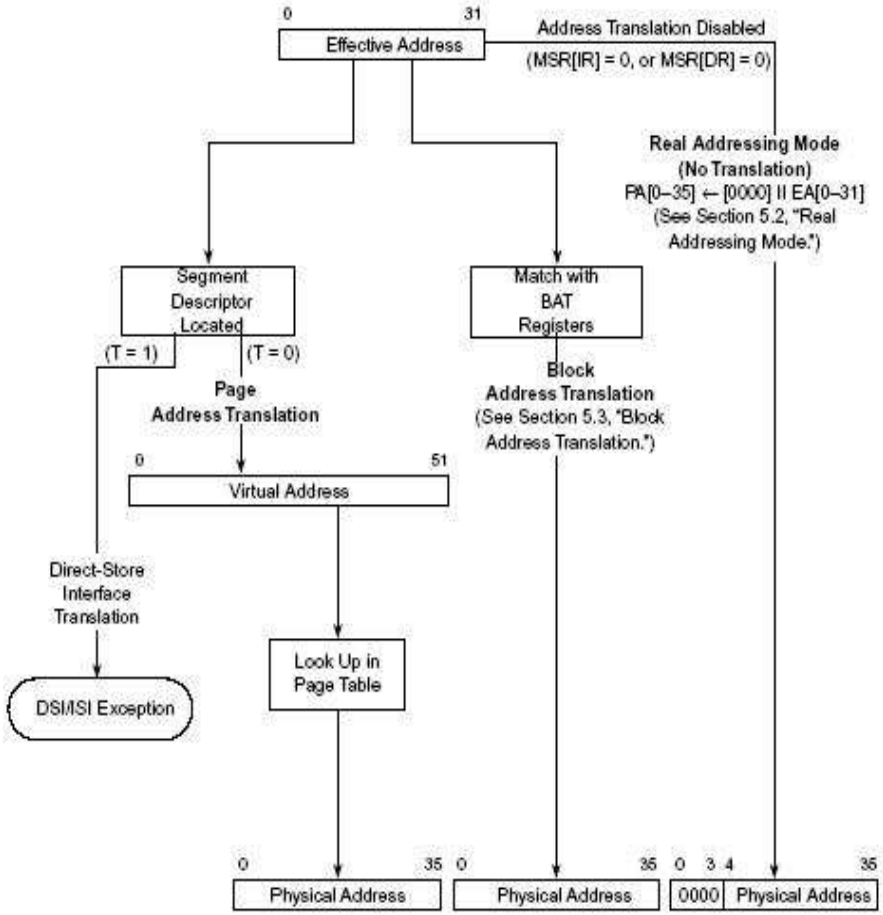
MMU OVERVIEW

- Separate DMMU and IMMU:
  - 128 entry two-way set associative ITLB and DTLB
  - 16 segment register
  - 8 register IBAT and DBAT
- Three types of addressing translations:
  - Real addressing mode
  - Block address translation
  - Page address translation
- Two sizes of phisical addresses:
  - 32-bit
  - 36-bit



REAL ADDRESSING MODE

- **Base**: the effective address is treated as the 32-bit physical address and is passed directly to the memory subsystem
- **Extended**: the 36-bit physical address is generated by having the system software add 4 leading zero's to the 32-bit effective address



BLOCK ADDRESS TRANSLATION

- For each access, an effective address is presented for page and block translation simultaneously. If a translation is found in both the TLB and the BAT array, the block address translation in the BAT array is used
- BAT provides a way to map ranges of effective addresses larger than a single page into contiguous areas of physical memory (128KB-256MB)
- BAT array is comprised of four entries used for instruction accesses and four entries used for data accesses.
- Each BAT array entry consists of a pair of BAT registers (BATU, BATL)
- If an effective address matches the corresponding BAT register field, the information in the BAT register is used to generate the physical address;

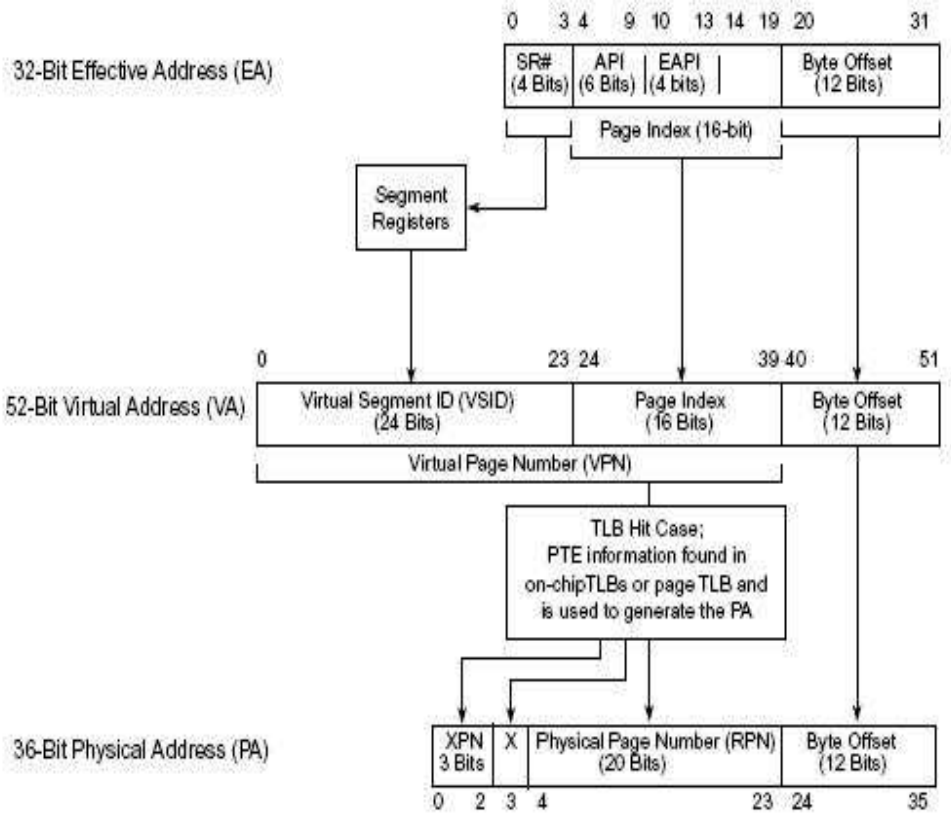
PAGE ADDRESS TRANSLATION

- In 32-bit implementations, the entire 4-Gbyte memory space is defined by sixteen 256-Mbyte segments.
- Segments are configured through the 16 segment registers.
- Segments subdivided into 4-Kbyte pages.
- This segmented memory model provides a way to map 4-Kbyte pages of effective addresses to 4-Kbyte pages in physical memory.
- Table search operations (to search for the PTE) performed in hardware or in software.
- If there is not a BAT hit, the page address translation proceeds in the following two steps:

    1. From effective address to the 52-bit virtual address;
    2. From virtual address to physical address.

PAGE ADDRESS TRANSLATION

- The 32-bit effective address is extended to a 52-bit virtual address by substituting 24 bits of upper address bits from the segment register.
- The 4 upper bits of the EA are used as an index into the segment register file.
- This 52-bit virtual address space is divided into 4-Kbyte pages, each of which can be mapped to a physical page.



ALTIVEC

- Altivec is Motorola implementation of SIMD: vector unit handles multiple pieces of data simultaneously in parallel with a single instruction
- Four 128-bit vector execution unit:
  - VIU1: executes AltiVec simple integer instructions
  - VIU2: executes AltiVec complex integer instructions
  - VPU: executes AltiVec permute instructions
  - VFPU: executes AltiVec floating-point instructions
- 32-entry, 128-bit vector register file (VRs)
- 16-entry, 128-bit renamed buffer

Unit	Latency
VIU1	1
VIU2	4
VFPU	4
VPU	2

ALTIVEC: VECTOR REGISTER

- Vector registers (VRs) are used as source and destination operands for AltiVec load, store, and computational instructions
- AltiVec's 128-bit wide vectors can be subdivided into:
  - 16 elements, where each element is either an 8-bit signed or unsigned integer, or an 8-bit character.
  - 8 elements, where each element is a 16-bit signed or unsigned integer
  - 4 elements, where each element is a either a 32-bit signed or unsigned integer, or a single precision (32-bit) IEEE floating-point number.

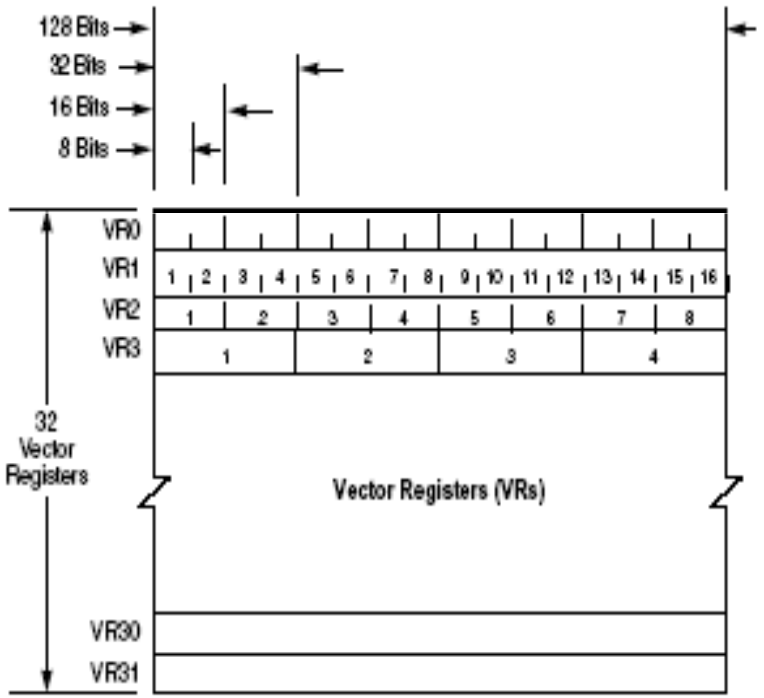
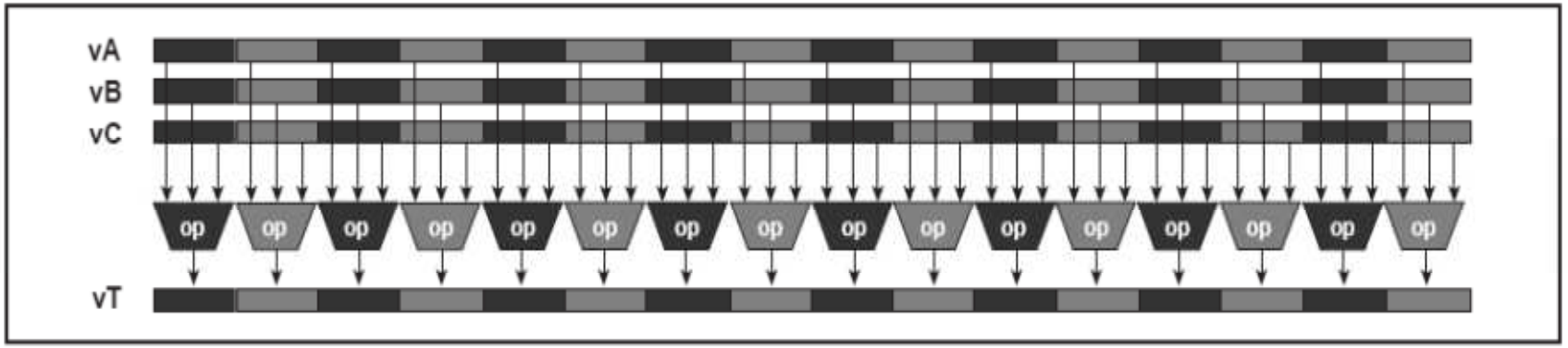


Figure 7-1. Vector Registers (VRs)

ALTIVEC: INSTRUCTIONS



- AltiVec technology adds 162 new “vector” instructions.
- Each AltiVec instruction specifies up to three source operands and a single destination operand.
- The target applications for AltiVec technology include: image and video processing systems, virtual reality, scientific array processing systems, network infrastructure such as Internet routers, etc...

AltiVec Operations

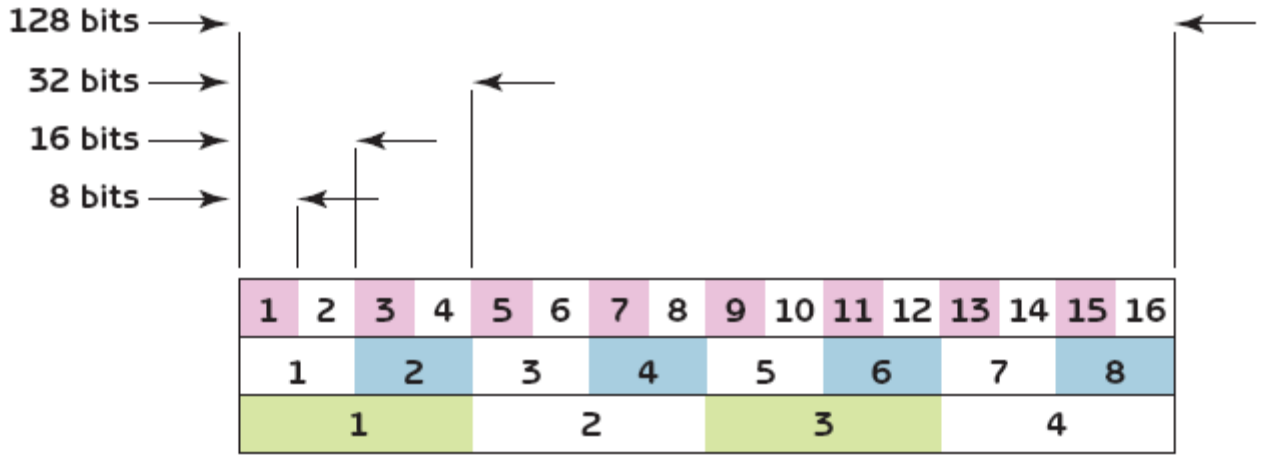
- 162 new PowerPC instructions -functionality similar to what is offered in the scalar units, just extrapolated into the SIMD domain.
  - new instructions for field permutation and formatting.
  - load/store instruction options for cache management.
  - instructions that control four data prefetch engines.
- The AltiVec vector unit never generates an exception.

AltiVec Data Types

vector { unsigned, signed, bool } { char, short, long }

vector float

vector pixel



Three alternative vector register data layouts

AltiVec Programming

- Recent versions of the GNU Compiler Collection, IBM Visual Age Compiler and other compilers provide intrinsics to access AltiVec instructions directly from C and C++ programs.
- The "vector" storage class is introduced to permit the declaration of native vector types, e.g., "vector unsigned char A;" declares a 128-bit vector variable named “A" containing sixteen 8-bit unsigned chars.
- AltiVec C extensions map into AltiVec instructions.
- For example, vec_add() maps into one of four AltiVec instructions (vaddubm, vadduhm, vadduwm, or vaddfp) depending upon the types of the arguments to vec_add().

Vec_Add

- Each element of a is added to the corresponding element of b. Each sum is placed in the corresponding element of d.

d = vec_add(**a**,**b**)

- Integer add:

```
n ← number of elements
do i=0 to n-1
d_i ← a_i + b_i
end
```

- Floating-point add:

```
do i=0 to 3
d_i ← a_i +_fp b_i
end
```