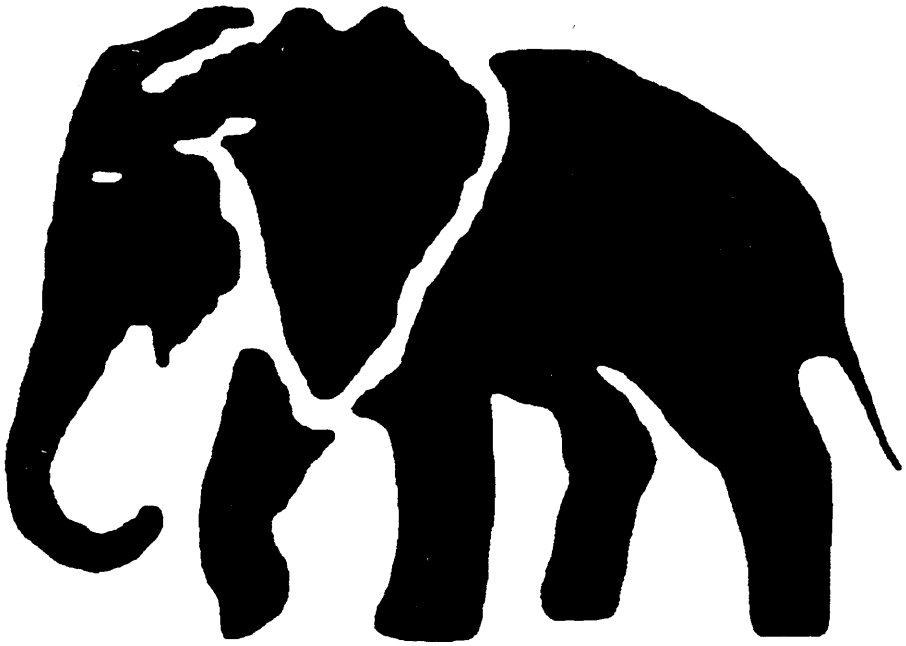


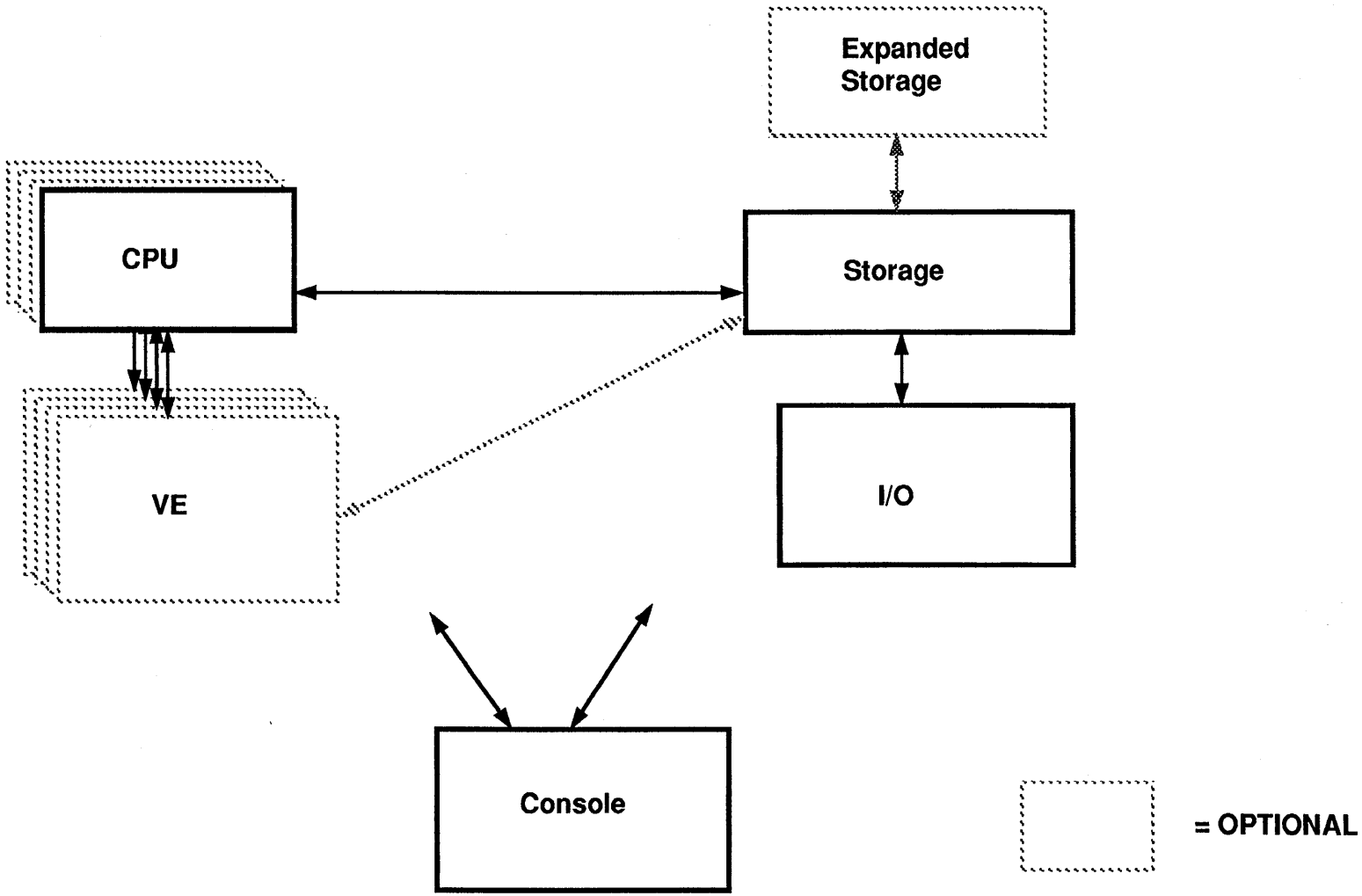
SONA HW Design Architecture

Part 1

Overview

370 Architecture Huge Picture

Rev. 2, 8/91



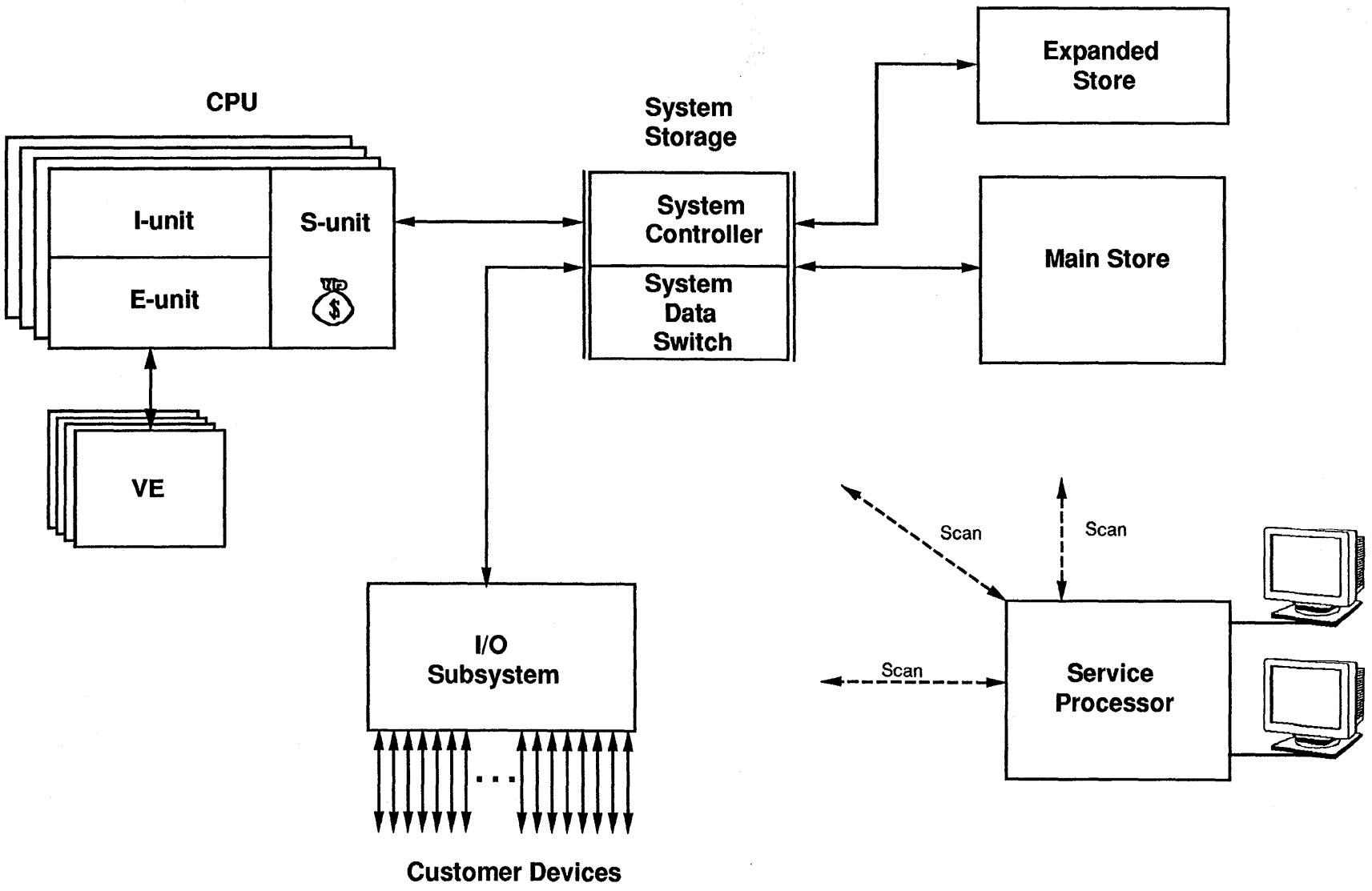
370 Architecture - Huge Picture

Required elements

1. CPU
2. Storage
3. I/O subsystem, including devices
4. Console

Optional elements

1. additional CPUs for multi-processing
2. Vector elements
3. Expanded storage



SONA Overview - SS System

- **CPU**
  - Up to 4 on a side (QP).
  - Each CPU includes I, E, and S units.
  - CPU fits on an MLG.

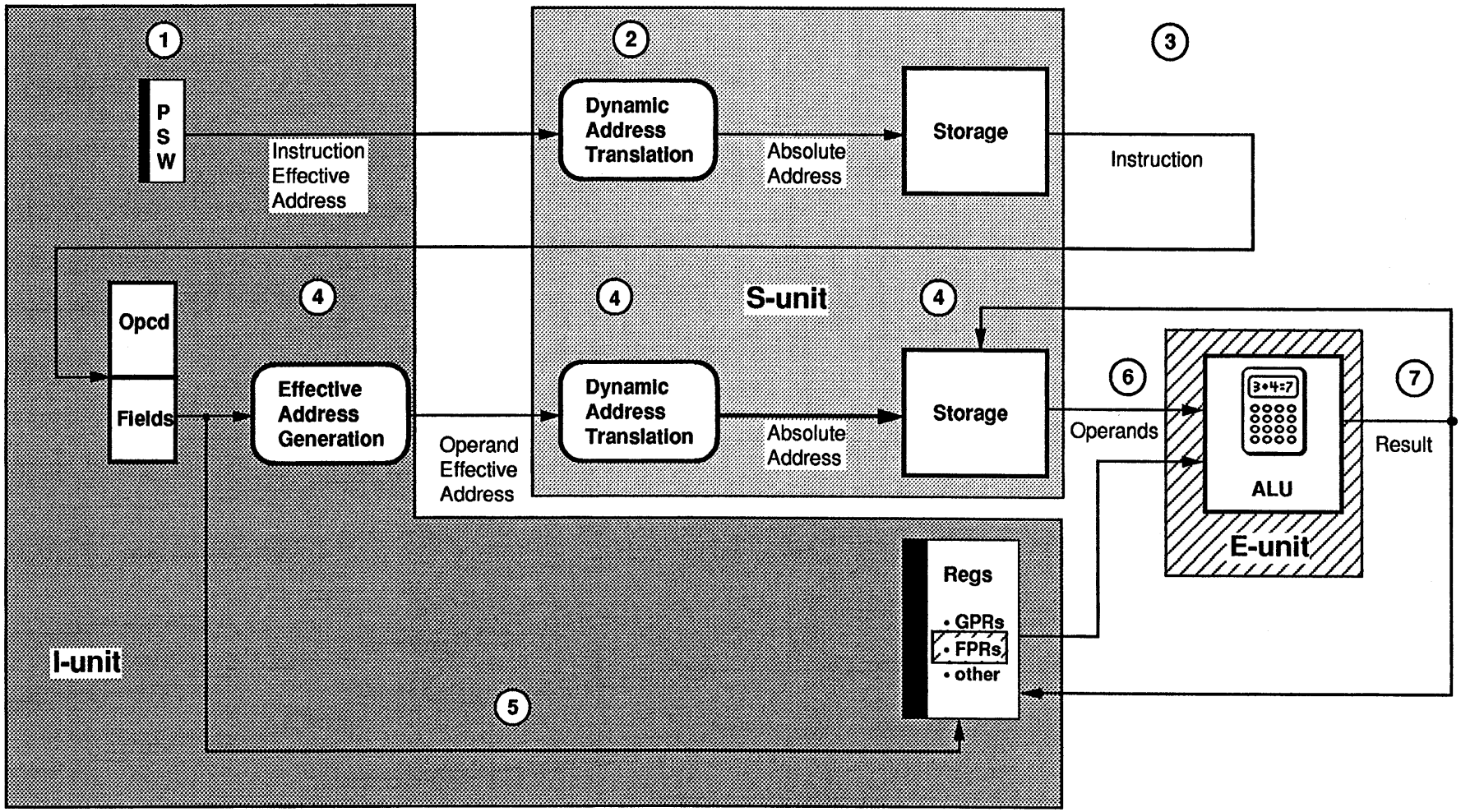
- **System storage**
  - Focal point for data traffic.
  - Includes Main Store and Expanded Store.
  - CPUs talk to System Storage, not to each other.
  - SC and SDS are each an MLG.
  - Main Store and Expanded store are implemented in ET technology on BLCs.

- **I/O Subsystem**
  - Gateway to the real world.
  - Linked to System Storage.
  - 1 MLG per IOP.
  - QDIH's (BLCs, ET) provide actual interfaces to devices.

- **Service Processor**
  - Stand alone computer system.
  - Has its own devices, including hard disks and terminals.
  - Communicates with mainframe via scan.
  - Implemented in ET technology on BLCs.

370 Architecture CPU Instruction Processing

Rev. 1, 5/91



370 Architecture - CPU Instruction Processing

1. Program Status Word (PSW) is starting point

- 64 bit register containing various parameters for instruction execution.
  - Current Instruction Address.
  - Dynamic Address Translation (DAT) mode: enables DAT (virtual addressing).
  - Condition Code (CC): result of prior operation. Used in conditional branch.
  - Key: 4 bits compared against storage key to verify that a storage access is OK.
  - Rupt/exception masks: enable/disable various interrupts.

2. Translate the instruction address

- if DAT is on, the address needs to be translated to an Absolute Address.

3. Fetch the instruction

- Includes an opcode and various fields *(described later).*

4. For storage accesses:

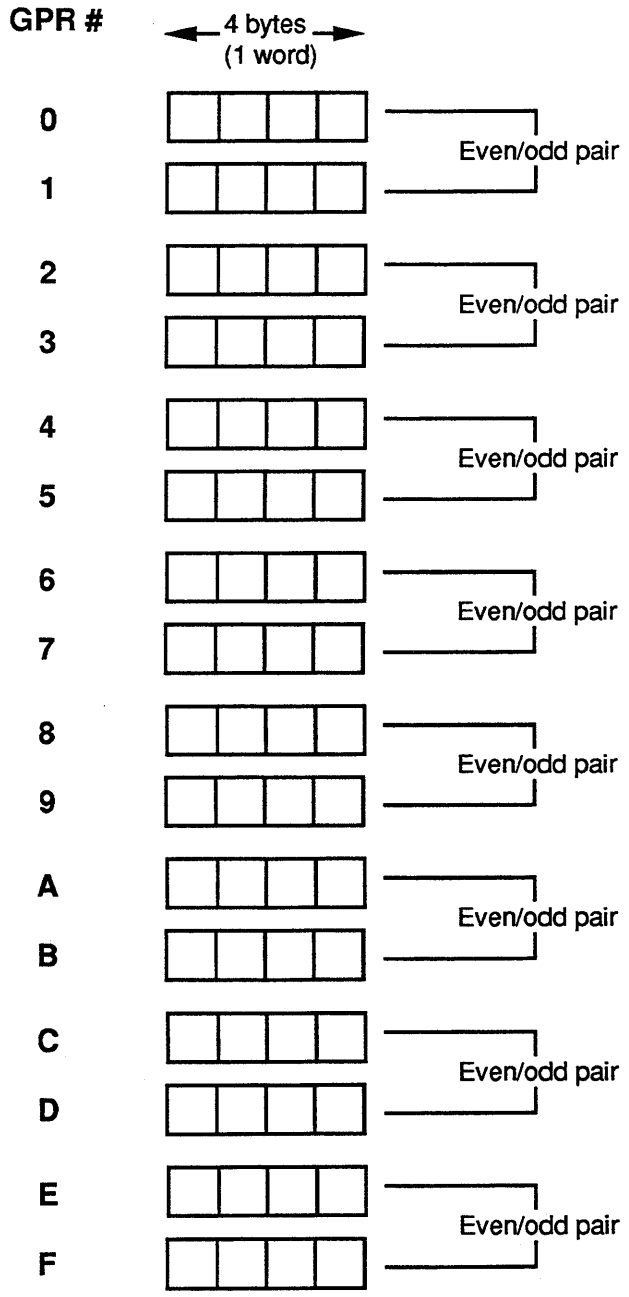
- Generate the Operand Effective Address. Fields in the instruction point to GPRs, which are used in Effective Address Generation (EAG).
- If DAT is enabled, translate the address to an Absolute Address.
- The Absolute Address is used to fetch the operand from storage.

5. For register operands:

- registers include:
  - General Purpose Registers: most common registers, used for most operations.
  - Floating Point Registers: used in floating point operations.
  - PSW: see above.
  - Control Registers: contain a variety of parameters, many used in DAT.
- a field in the instruction points to the register.

6. Send the operands to the ALU for processing.

7. Store the results away in either storage or a register.



General Purpose Registers

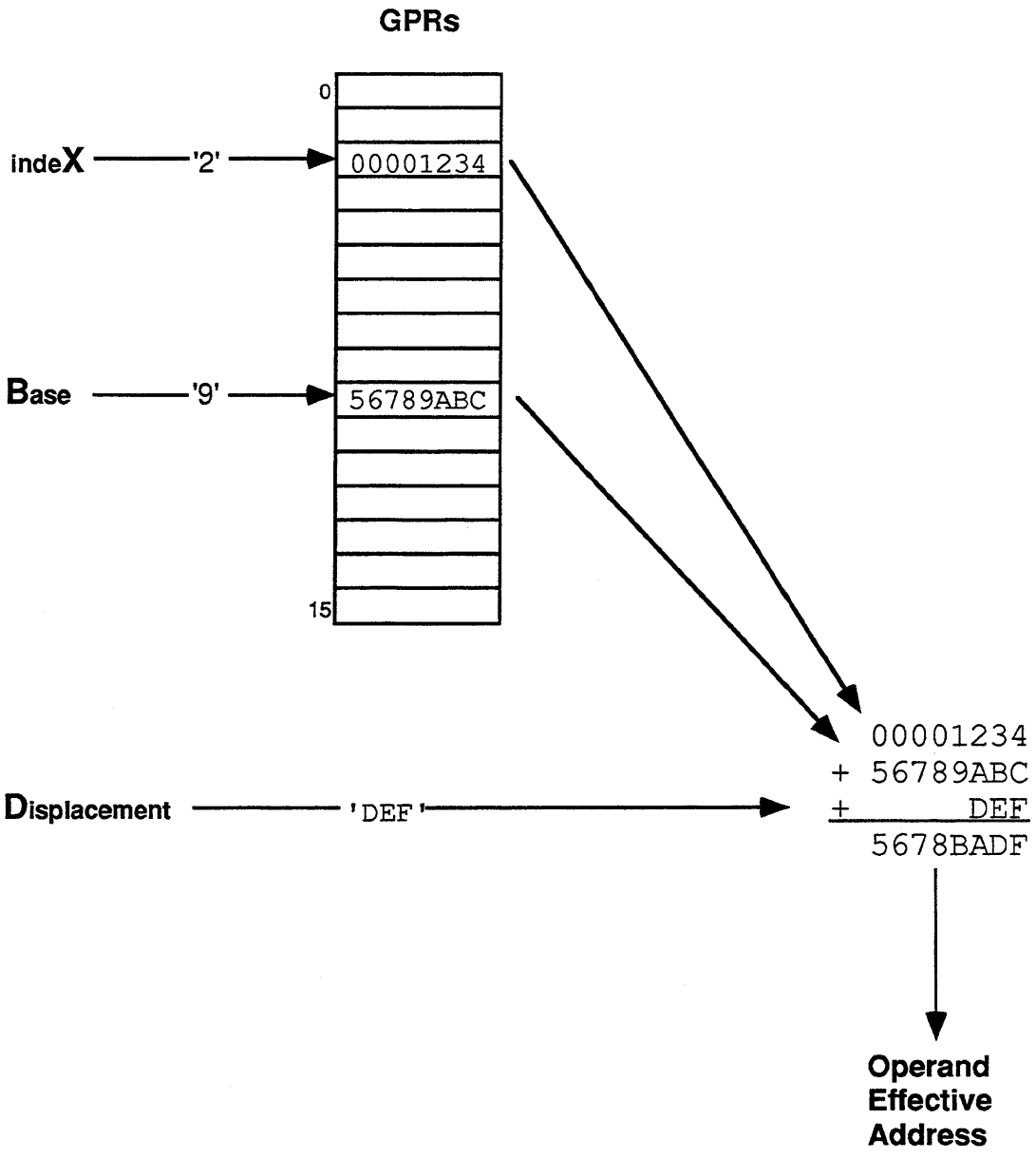
- **General Purpose Registers.**
  - Also called just General Registers in POO.

- **Focal point for a lot of processing.**
  - Contain general data results.
  - Also can contain addresses (see EAG).

- **16 of them.**
  - Four bit field indicates which register to use.

- **Four bytes wide = 1 word.**

- **Can be addressed in even/odd pairs to do double-word operations.**
  - Register field points to even register. Second (odd) register is implied by opcode.



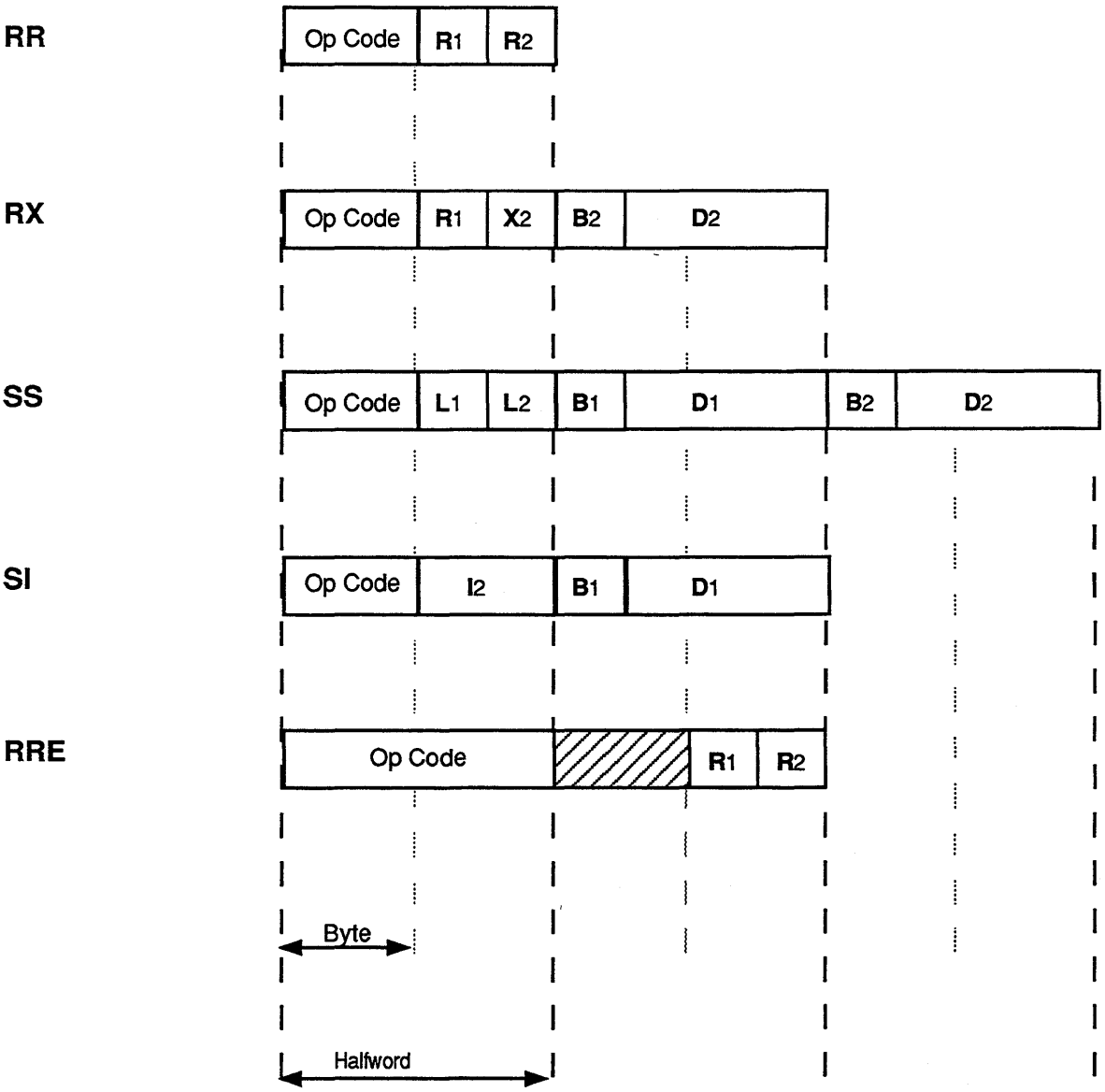
Effective Address Generation

- **Uses two GPRs, indeX and Base.**
  - Index GPR pointed to by X field of the instruction.
  - Base GPR pointed to by B field of the instruction.

- **Adds them in with 12 bit (right aligned) Displacement.**
  - Displacement is also a field of the instruction.

Sample Instruction Formats

Rev. 1, 5/91



Sample Instruction Formats

- **RR**
  - Operates on two registers.
  - Typically, both registers are inputs to the operation and the result is stored in R1.

- **RX**
  - Operates on a register (R1) and a storage location (X2, B2, D2).
  - Typically, both operands are inputs to the operation and the result is stored in R1.
  - X2, B2, and D2 are inputs to EAG to generate the address of the first byte of data.

- **S S**
  - Storage to storage.
  - Typically, two fields of storage data form the operands.
  - Results are usually stored to the first operand.
  - The L fields indicate the length of each operand.
  - EAG (without an index) points to the first byte in each field.

- **S I**
  - Storage-Immediate.
  - One operand is storage, the other is a field in the instruction itself.

- **xxE (e.g. RRE)**
  - Extended (2 byte) opcode.
  - First byte points to a "family" of related opcodes.

- **Note:**
  - Instructions are 1, 2, or 3 half-words. Must be half-word aligned.
  - Register addresses are nibbles.
  - Displacement is 12 bits.

Exercise

PSW	00	09	00	00	00	00	10	00

GPR 1	00	00	10	00
GPR 2	00	00	00	00
GPR 3	FF	0F	FF	FF
GPR 4	00	00	00	01
GPR 6	00	00	00	01

Storage

0FF0	49	32	00	90
0FF4	37	FC	A0	03
0FF8	54	9C	FF	FF
0FFC	47	32	FF	F3
1000	54	32	10	08
1004	50	32	10	08
1008	47	F2	10	00
100C	1A	44	1B	56
1010	47	32	10	0C
1014	50	42	20	00
1018	47	F2	10	18
101C	FC	FF	FC	FF
1020	21	32	48	AB
1024	ED	B9	45	51

Architecture Summary

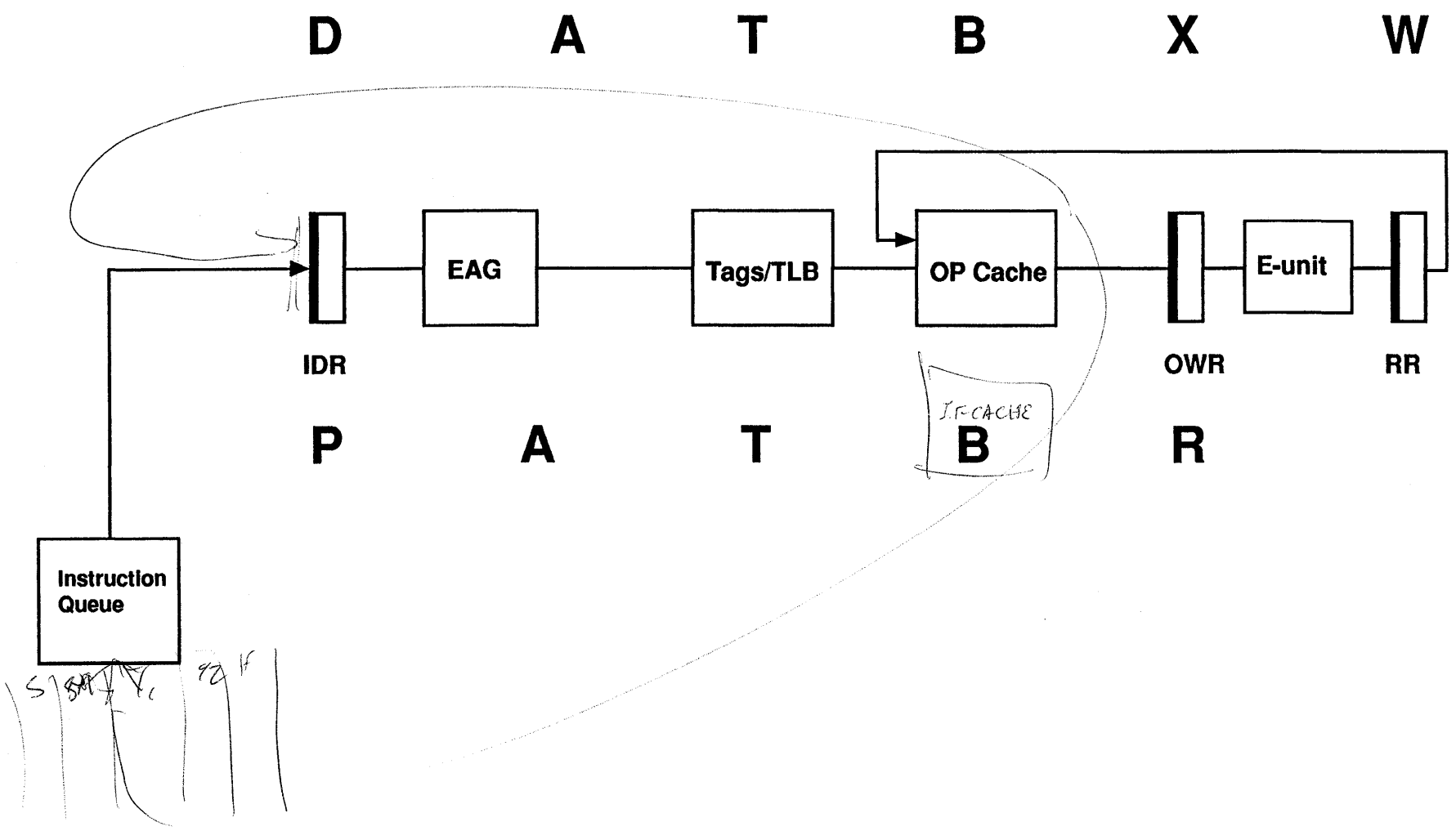
Key points of architecture.

- PSW has Instruction Address, Has condition code,
- 16 4 byte GPRs
- EA6 uses (index, base, disp) to gen OP address
- 2,4,6 bytes length instrct w 1-2 byte opcodes
- Lock reference Reg and/or Storage
-

SONA Pipeline Overview

Rev. 1, 5/91

TSR



SONA Pipeline Overview

SONA Pipeline

- **D** Decode instruction to generate controls. Do Effective Address Generation.
- **A** Address sent to the S-unit.
- **T** TAG/TLB access.
- **B** Buffer (cache) access.
- **X** eXecute the instruction.
- **W** Write the results back.

Notes

- This pipeline is interlocked.
- Two cycles are not shown - they're I-unit constructs and are generally transparent to the rest of the machine. They're not part of the "official" pipe (at least in my view).
  - **C** Control store access on second and subsequent flows. In front of D.
  - **Z** When data is really written to the GPRs and other registers. After W.

Key Registers

Instruction Data Register - holds 4 bytes of instruction.

Operand Word Register - contains 8 bytes (doubleword) of operand data.

Result Register - contains 8 bytes of operation results.

S-unit Pipeline

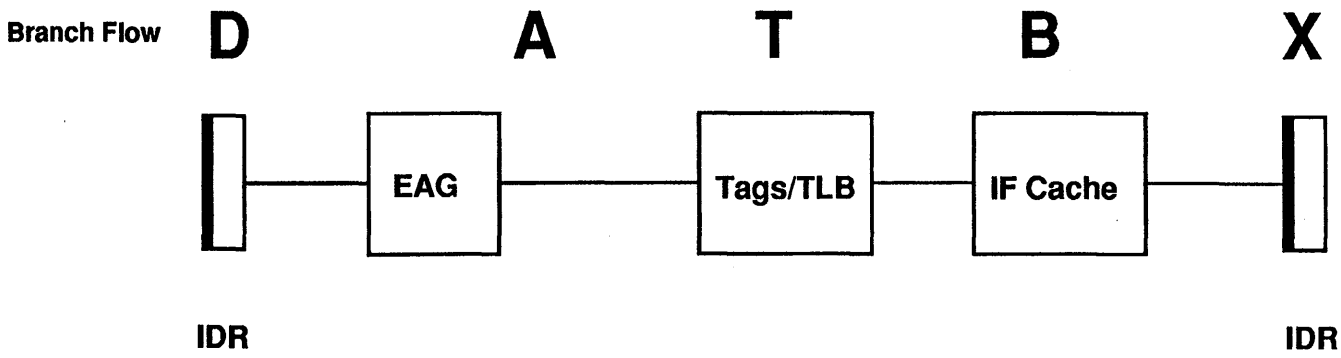
- **P** Initial priority cycle.
- **A** Address selection (based on final priority)
- **T** TAG/TLB access.
- **B** Buffer (cache) access.
- **R** Data (results) clocked into OWR.

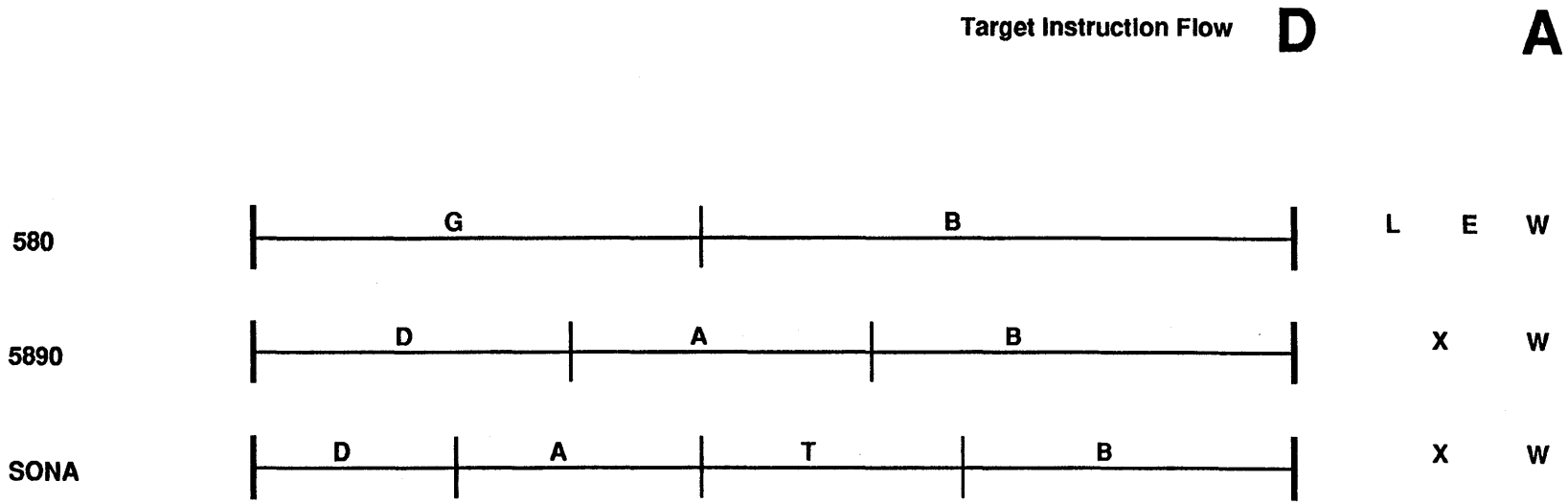
Note

- The S-unit pipeline is free-running.

- This page intentionally left blank -

- This page intentionally left blank -





Branches

Branch Processing

- Goes through EAG like any other RX instruction. *(D,A cycle)*
- Address sent to TAGs/TLB and Buffer. *(T, B cycles)*
  * Note: non-branch instructions will access ***OP*** Tags and buffers, whereas a branch accesses ***IF*** Tags and Buffer. The timing of these two accesses is the same.
- New instruction data is loaded. *(X cycle)*
  * Note: non-branch instructions will load ***OP*** data into the ***OWR***, whereas a branch will load ***IF*** data into the ***IDR***.
- Since D cycle of target lines up with X cycle of branch, there's a 3 cycle *branch penalty* incurred on taken branches.

Branches and performance

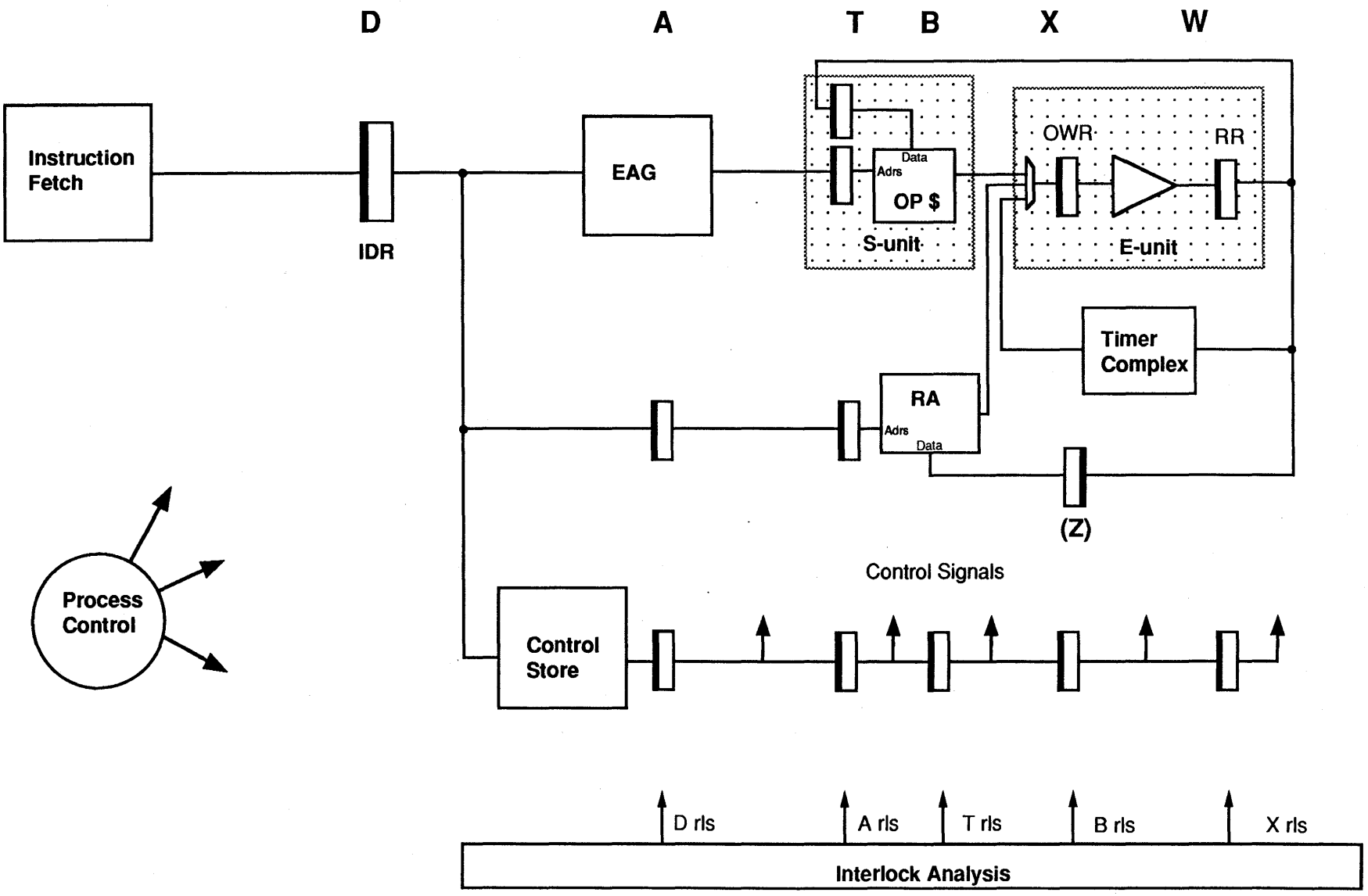
- The machine cycle time can't be faster than the raw branch path delay (IDR⇒EAG⇒TAG/TLB/Cache⇒IDR) divided by the number of cycles in this path.
  * SONA divides this over 4 cycles - DATB.
  * 5890 divided this over 3 cycles - DAB.
  * 580 divided if over 2 cycles - GB.

- the down side of having more cycles is:
  * _____
  * _____
  * _____

I-unit

I-unit Basic Blocks

Rev. 2, 9/91



Basic Blocks

- **Instruction Fetch**
  - Maintains a queue of instructions.
  - Uses queue to keep ____ filled.

- **Instruction Data Register (IDR)**
  - Primary D-cycle platform.
  - Holds 4 bytes of the current instruction.

- **Effective Address Generation (EAG)**
  - Adds indeX, Base, and Displacement to generate the operand effective address.

- **Register Array (RA)**
  - A variety of registers, including the GPRs.

- **Timer Complex**
  - Includes a Time of Day clock and facilities to count time intervals (for time slicing).

- **Control Store**
  - Generates control points for the pipe.
  - Combination of µcode and hard-wired control.

- **Interlock Analysis**

  _____

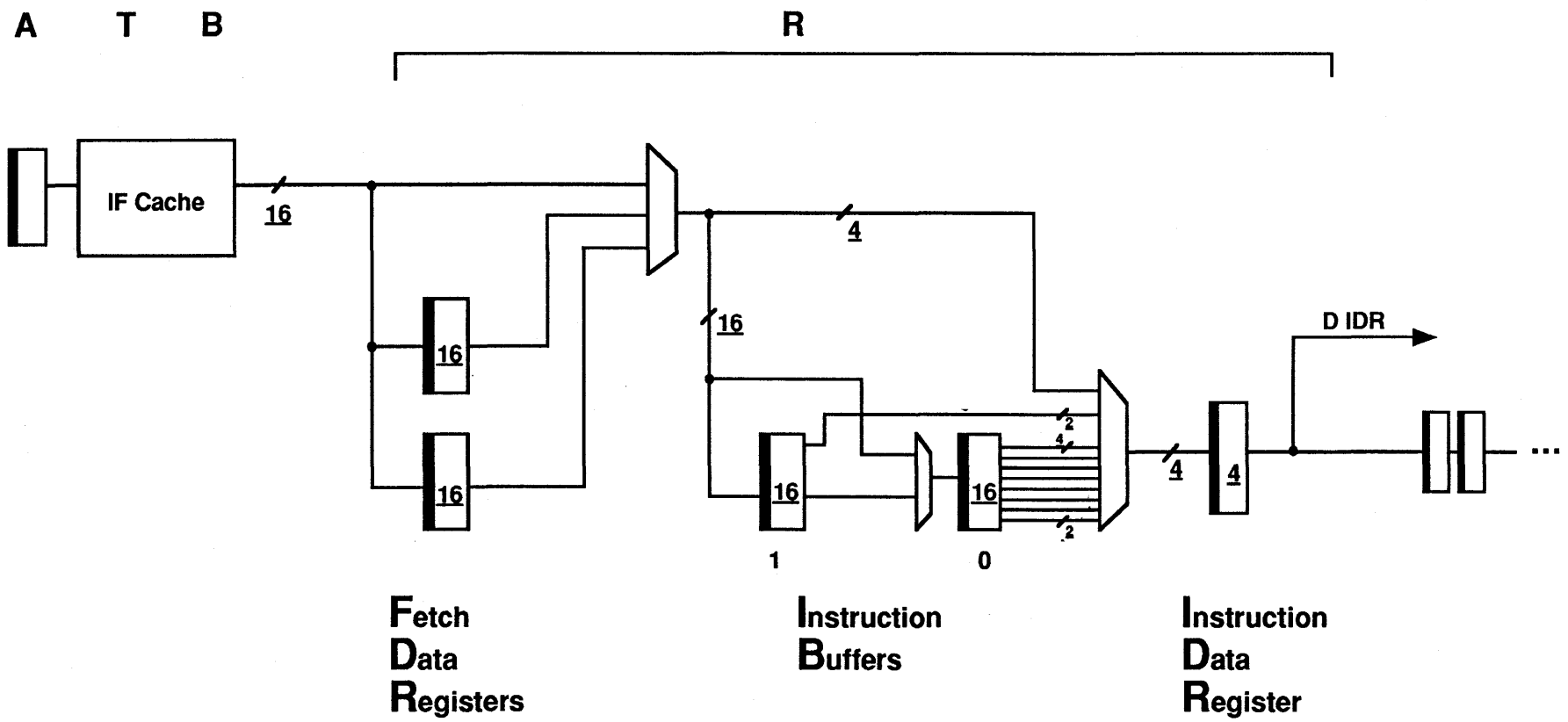
  _____

- **Process Control**
  - State machine to control process switching (interrupts).

- **Non I-unit stuff**
  - OP cache (a.k.a. buffer) in S-unit.
  - E-unit, including OWR and RR.

I-fetch Data Paths

Rev. 1, 5/91



I-fetch Data Paths

I-fetch charter

Function of paths (excluding FDRs)

- IF cache:
  * Load IDR
  * Load IB0-IB1
- IB0:
  *
  * 16 byte Queue into IDR 8 halfword Q
- IB1:
  * Replenish IB0
  * To 2 byte IB1 from Bottom 2 of IDR
  * acts as shifter.

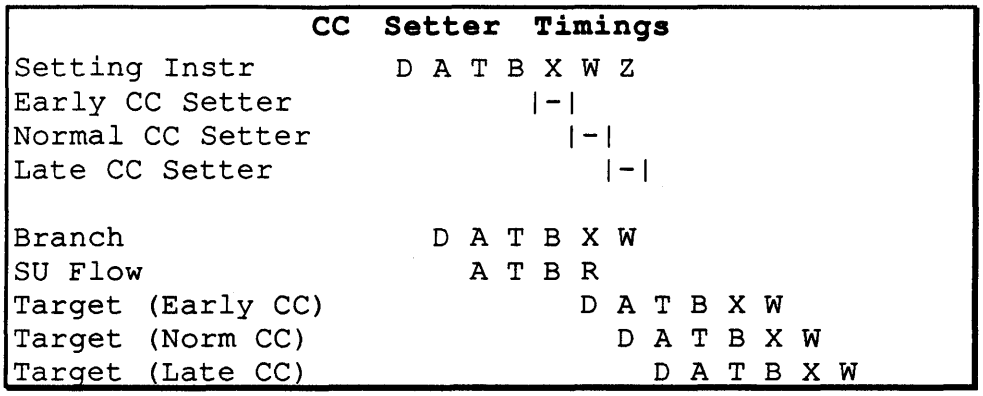
- **Note:**
  - the IDR is only 4 bytes. For 6 byte instructions, we start out with the first 4 bytes, which are enough to generate the first address. Once this is done, the third HW overclocks the 2nd HW, allowing us to then generate the second address.

- This page intentionally left blank -

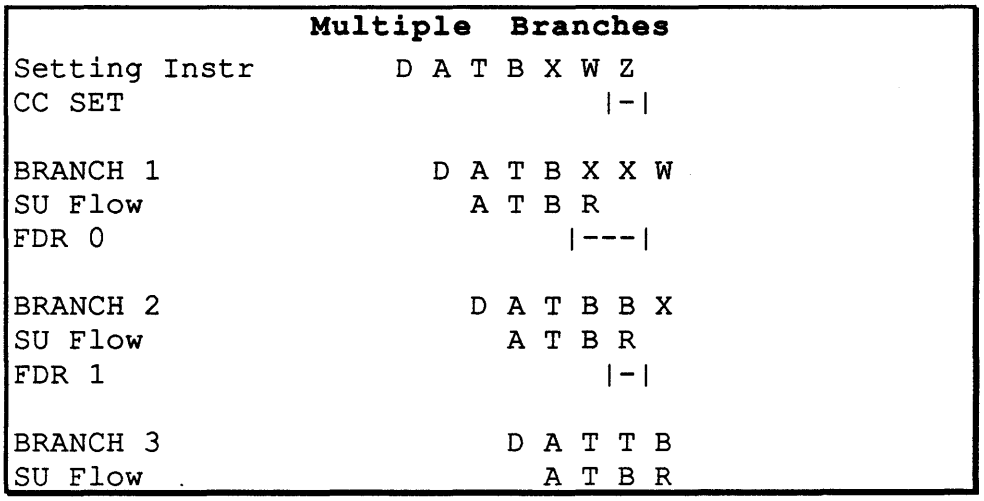
I-fetch Data Paths (cont.)

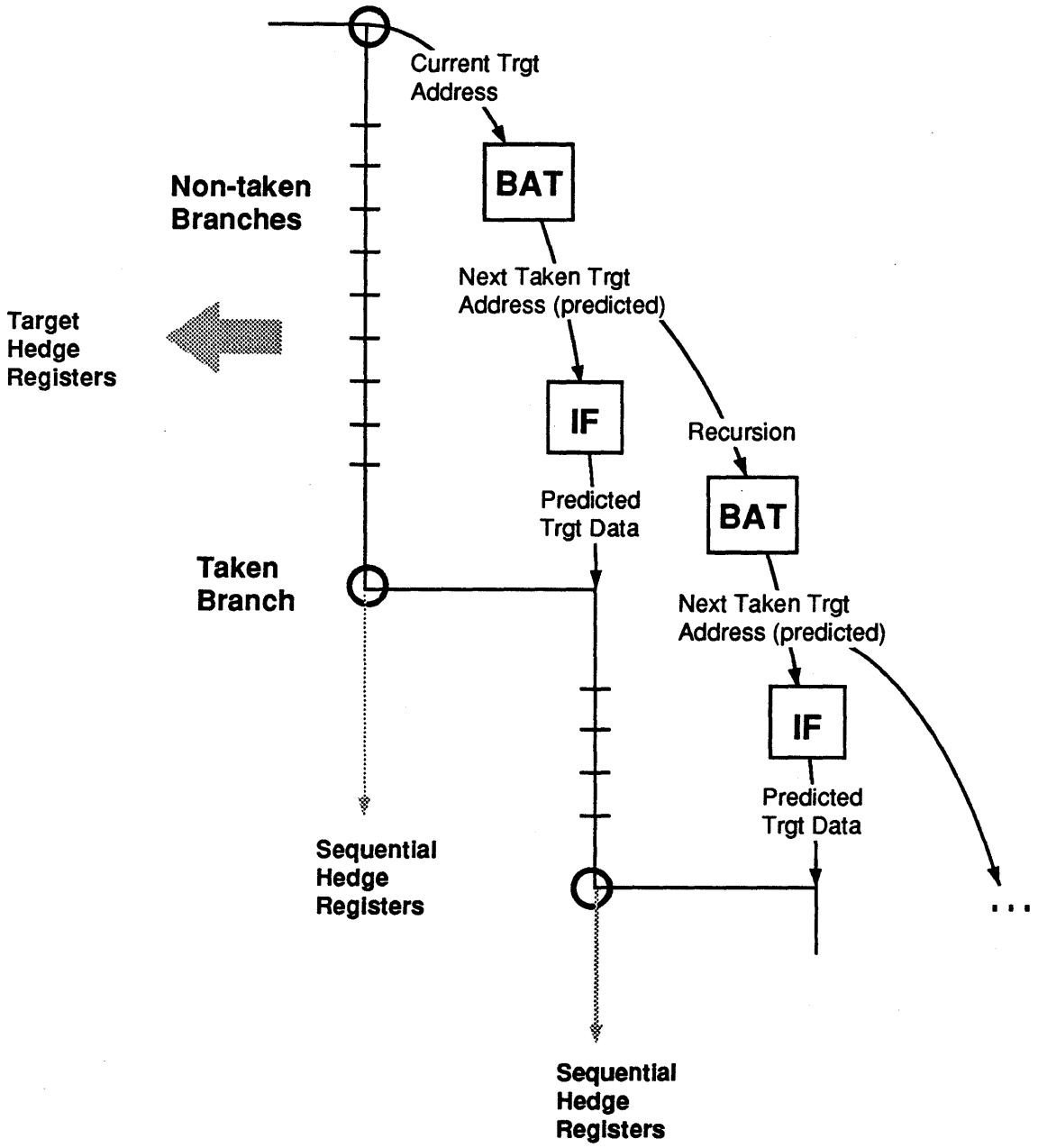
Branch Processing

- Instructions that set the CC will do so in 1 of 3 cycles:
  * Early Setters: X
  * Normal Setters: W
  * Late Setters: Z
- Subsequent branch instructions can't make the *branch decision* (what to load into the IDR) until the CC has been set. Thus, the branch penalty is increased for Normal and Late CC setters.



- Since the S-unit is free-running, it may return the data before the branch decision has been resolved. Until then, the data needs to be stored somewhere.
  * I-fetch provides Fetch Data Registers (16 bytes each) to hold target instruction data until the branch is resolved.
  * Two FDRs provide enough for worst case (late CC setter followed by 3 branches).

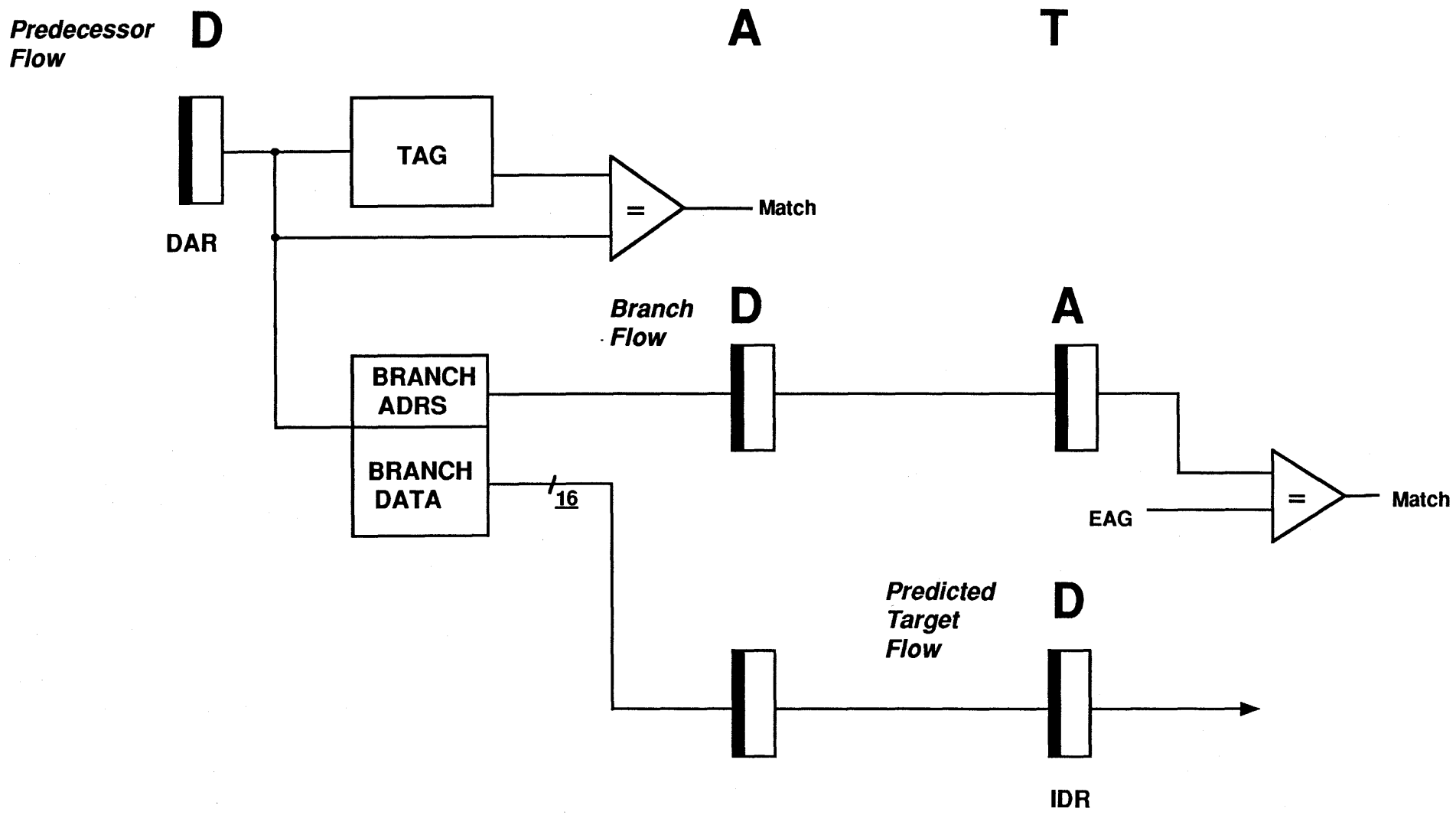




Branch Target Buffer

Rev. 1, 5/91

TSR



Branch Address Table Concept

Goal: Whenever a branch is encountered, it takes a while to fetch the target data and resolve the branch decision. In the meantime you'd like to keep the pipe busy. In the current design IF guesses that the branch will be _____ and fills these pipeflows with _____.

The goal is to improve this guess. To do so you need to:

1. Have some way to predict which branches will be taken, then fill the pipe with the target stream following these branches. This requires that you ...
2. Prefetch the target data so it's ready for execution. Since it takes a while to fetch data from the cache, the prediction needs to be made well ahead of time so you have a chance to do this prefetching.

Implementation:

- The Branch Address Table is addressed by the Target Address of taken branches.
- The BAT contains the Target Address of the next taken branch. It also contains a count of how many branches are not taken before the next taken branch. Both of these fields are written into the BAT whenever a branch is first taken (i.e. ¬predicted branch).
- In words: the last time I branched to this location, the next taken branch was x branches later, and it branched to location y.
- Using the Predicted Target Address, prefetch the target data from the IF cache and keep it handy.
- Keep track of non-taken branches. When you get to the one that should be taken, let the next pipeflow use the prefetched target data instead of sequential data.
- The branch still needs to be processed and the branch decision checked to verify that the prediction was correct. Similarly, the generated target address is compared with the predicted address to make sure the address is correct.

Hedge Registers:

- On branches that are predicted to be not-taken, the branch flow still fetches the data and puts it in a Target Hedge Register, pending resolution of the branch. If it ends up being taken after all, this data can be loaded into the IDR to start up the next stream. Target Hedge Registers function just like _____ do in the current design.
- On branches that are predicted to be taken, save whatever data you have queued up for the sequential stream in a Sequential Hedge Register. That way if the prediction is wrong you can quickly restart the sequential stream.
- In either case, if you predict wrong you have to cancel the flows that got fired up. This is the same as it is today.
- This structure allows recursion. The Predicted Next Target Address can be used to address the BAT to get the follow-on Target Address, and so on. Each of these follow-on addresses can be sent to the IF cache to get the corresponding data, thereby building up a queue of target data (along with the associated addresses and NTCs).
- The current plan is 1 level of recursion (i.e. fetch next 2 targets) for the data and 3 levels of recursion for the address and the Not Taken Count.

Branch Target Buffer
(obsolete)

Goal: For each branch decision, correctly guess what the decision will be and have the data ready in time for the first D-cycle after the branch.

Approach: Keep track of taken branches and guess that they'll be taken again.

Implementation:

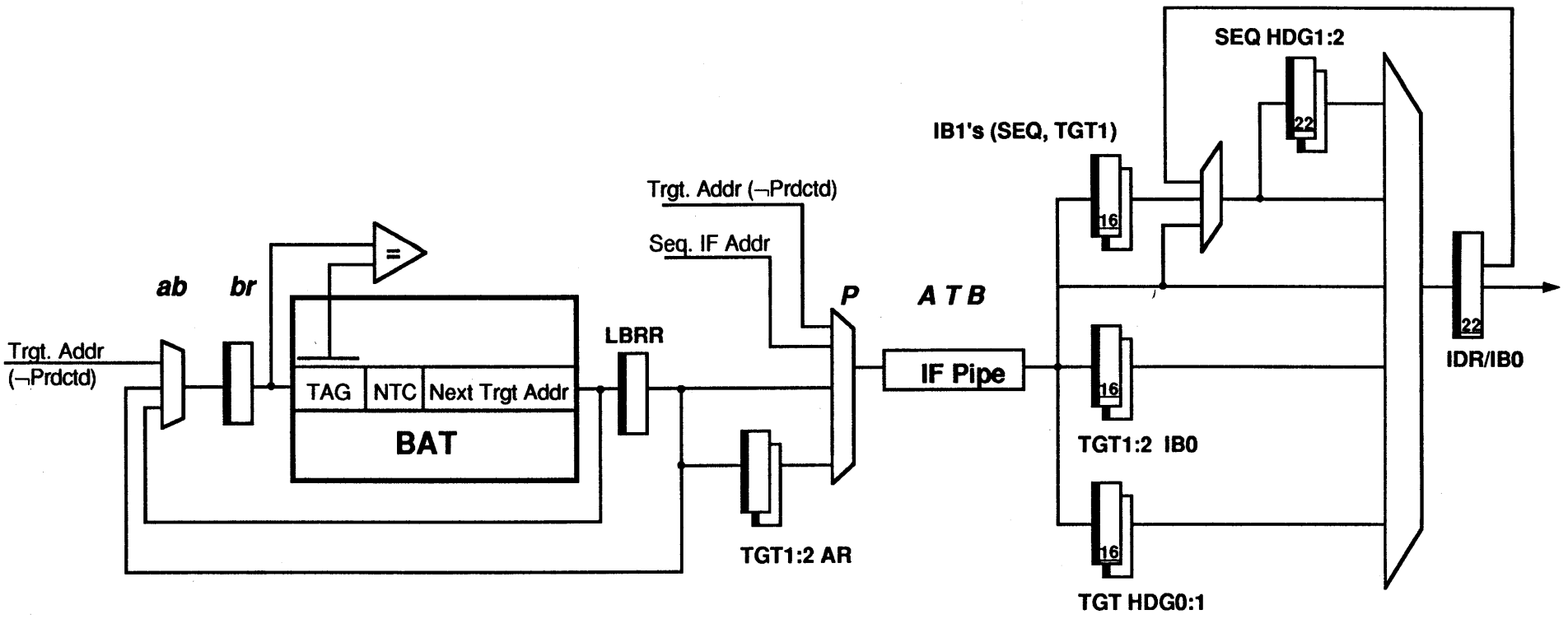
- Keys off of predecessor instruction address (instr. before branch) for timing reasons.
- In words: *Last time I was at this instruction address, the next instruction was a taken branch, so I'll assume that's what is going to happen this time.*
- The BTB saves the instruction data fetched from the previous time around, and loads it into the IDR to start processing.
- A "complete" BTB would have an entry for _____. Instead, only a portion of this conceptually huge address space is saved using standard caching techniques.
  * 256 sets x 2 associativities.
  * Addressed by low order bits of predecessor instruction address.
  * Remaining bits stored in TAGs and matched against.
  * The "data" includes the branch data (i.e. target instruction) to be loaded into the IDR, plus the target address, which has two uses.
    - The predicted target address is compared with the calculated target address (from EAG) to provide early detection of an incorrect prediction.
    - It provides an early copy of the target address to access the BTB, as the target instruction could be a predecessor itself.
  * This data can then be loaded into the IDR the cycle after the branch D-cycle.
  * The branch processing continues as usual to allow verification of the BTB data:
    - verify that it is, indeed, a branch.
    - verify that the branch is taken.
    - check that the branch address is the same as the predicted address.
    - compare the data fetched by the branch flow with the data taken from the BTB.
  * If any of these checks detects a problem, the pipeflows spawned from the BTB data are cancelled and the IDR is re-loaded with the correct instruction.

Branch Target Buffer Timing

```
Predecessor Instr    D A T B X W
Branch                 D A T B X W
SU Target Fetch        P A T B R
Branch address chk      |-|
Branch decode chk       |-|
Branch decision chk         |-|
Data mismatch chk             |-|
1st predicted instr      D A T B X W
```

BAT Design

Rev. 1, 3/92



Branch Address Table Design

BAT Design

- 4K buffer, 2-way set-associative
- Addressed by Target Address 20:30
- TAGs include:
  * Target Address 12:19
  * Domain #
  * Guest/Host bit

- Access cycles include:
  * ab - address BAT cycle
  * br - BAT read cycle. On writes this becomes bw.

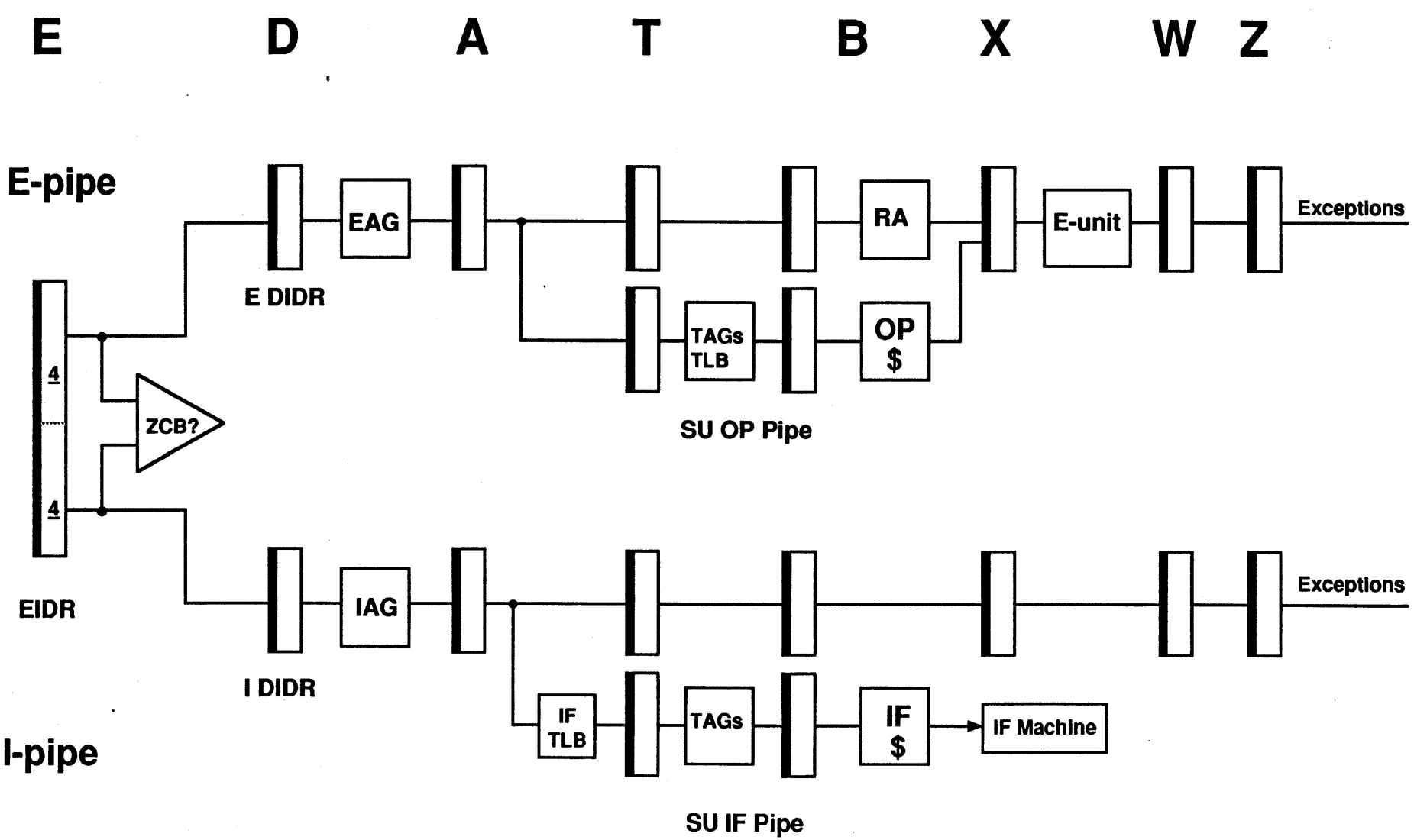
- Contents include a prediction (based on last time around) for:
  * Next Target Address 1:30
  * Non-taken count 0:3
    - Number of non-taken branches before next taken branch.
    - A count of F means invalid. This saves having a valid bit.

- The BAT can be accessed recursively in consecutive cycles. If this recursion gets interrupted it can be restarted from the LBRR (Last BAT Recursive Read). Predicted addresses can be saved in TGT1:2 and LBRR, with a 4th address on the BAT outputs.

IF Data Paths

- TGT1:2 AR are each sent down the IF pipe to prefetch target data, which is saved in their respective IB0s.
- In addition, the second 16 bytes for TGT1 are prefetched and stored in TGT1 IB1.
- When the predicted branch is encountered, the data from the corresponding TGT IB0 is loaded into the IDR/IB0 and processing commences on it. For TGT1 this can be further replenished from TGT1 IB1.
- Meanwhile, the top 22 unused bytes of the IDR/IB0 and SEQ IB1 are saved in a Sequential Hedge Register, pending verification that the branch is indeed taken.
- Sometimes the new target stream will immediately encounter another branch which is predicted to be taken. This may occur before the original branch is resolved. To handle this a second Sequential Hedge register is provided to save the just loaded IDR/IB0 contents. Once the branch decisions are resolved these hedges will either be cancelled or one of them will be re-loaded into the IDR/IB0.
- The IDR and IB0 from the current design are combined into one 22-byte register. The top bytes are used as the IDR, and the whole thing is viewed as IB0.
  - For Zero Cycle Branches you always want to have 8 bytes or more in the IDR. Thus, when the count falls to 8 you know you want to load in 16 bytes from IB1 next cycle. Meanwhile, at least 2 of the current bytes will be consumed by the pipe, so room is needed for 6+16=22 bytes.
  - Since the Sequential Hedge Registers are fed by the same shifter bus as the IDR/IB0, making them also 22 bytes each simplifies things.

Zero Cycle Branch
Rev. 1, 3/92



Zero Cycle Branch

Basic Concept

- For certain cases of certain branch instructions (BC, BCR), execute the branches in parallel with the previous instruction.

Basic Implementation

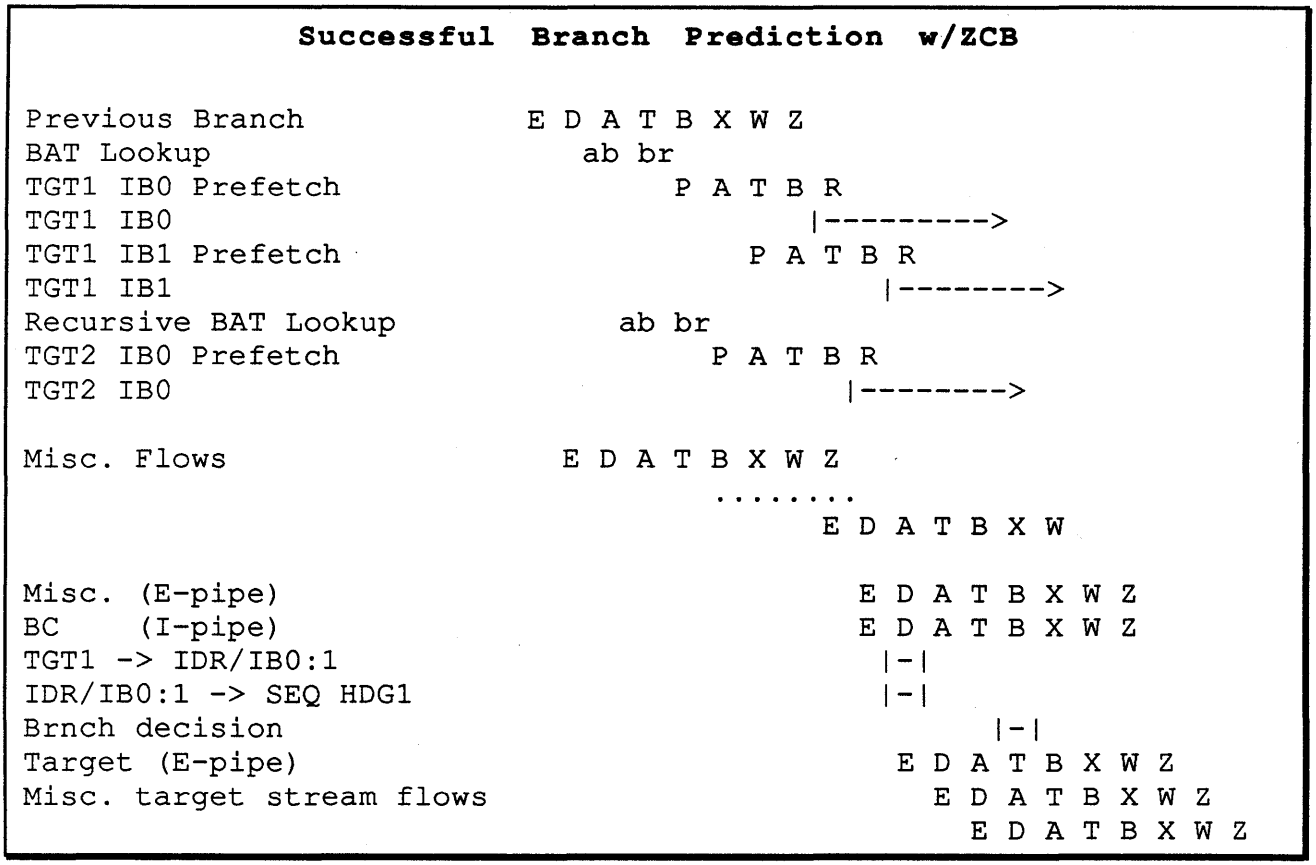
- In addition to the original pipe (now called the E-pipe since it can use the E-unit), a second parallel pipeline (I-pipe) is created. This pipeline is dedicated to executing BC/BCR and has minimal facilities. Specifically, it has:
  - Instruction Address Generation
    * Generates the target address
    * Only has a 2 port adder. Either the base or index must be zero to do ZCB on a BC.
    * Has a dedicated selector to read out the base/index from the EAG GPRs.
    * No EGI bypass provided.
  - SU IF Interface
    * Interface to the IF pipe to fetch the target data.
    * SU IF pipe now has a 4 entry TLB. On TLB match (95% of IF's) this allows the target fetch to complete w/o the OP pipeline.
    * On TLB miss (done in the A-cycle) a traditional IF TLB validate is initiated a cycle later than normal, requiring the OP pipe.
  - Staging of address, opcode, misc. stuff
    * Used for exceptions, PER, Address Compare, STIS.

- An Extra (or Eligibility) cycle is added to the front end of the pipe.
  - The EIDR is examined to determine if the first two instructions can be "paired". A number of conditions must be met. A partial list includes:
    * The second instruction is a BC or BCR. If a BC, one of X or B must be zero.
    * Certain first instructions can't be paired with a BC.
    * No EGI.
  - This extra cycle also allows the IDR to be latched before being distributed to lots of DIDR copies. In the current design this distribution is done directly from the IF cache, creating some physical design problems.

- If pairing occurs (between the first instruction and the subsequent BC/BCR), the two instructions will proceed down their respective pipes in lockstep with each other. That is, if either pipe interlocks, they both will interlock.

- If all goes well (e.g. BC/BCR is eligible, IF TLB match) the BC/BCR can be executed using only the I-pipe, effectively costing zero cycles.

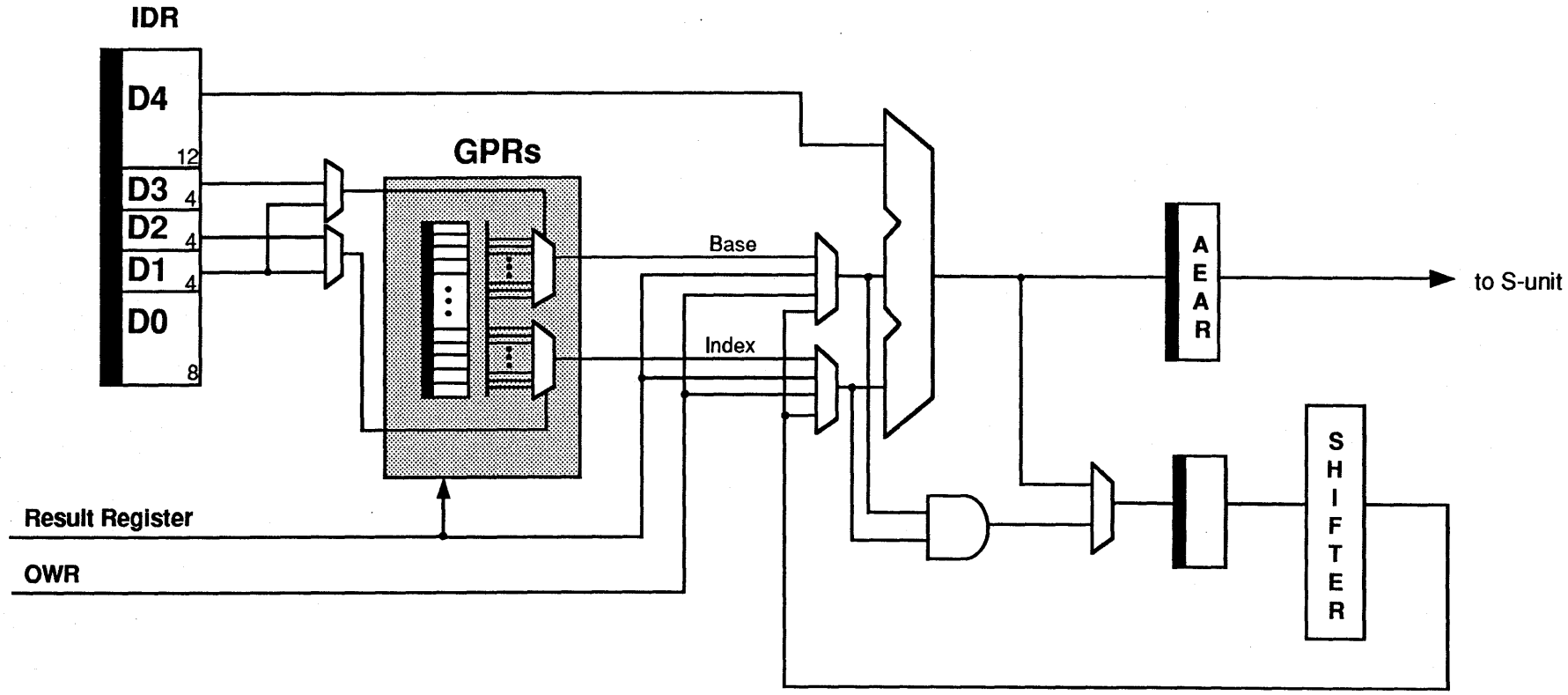
Sample Timing



- This page intentionally left blank -

EAG Data Paths

Rev. 1, 5/91



EAG Data Paths

- **Effective Address Generation**
  - EAG complex maintains a copy of the GPRs.
  - X and B fields used to select index and base GPRs, respectively.
  - Index and base GPRs fed to a 3 port adder, along with the displacement field.
  - Result is put into the AEAR which sends it to the S-unit.
  - Note the path from the RR to update the GPRs at the end of the W-cycle.

- **EGI interlock**
  - If a prior instruction is modifying the B or X GPRs, EAG must wait for it to be updated from the Result Register.
  - Can be a substantial performance penalty.

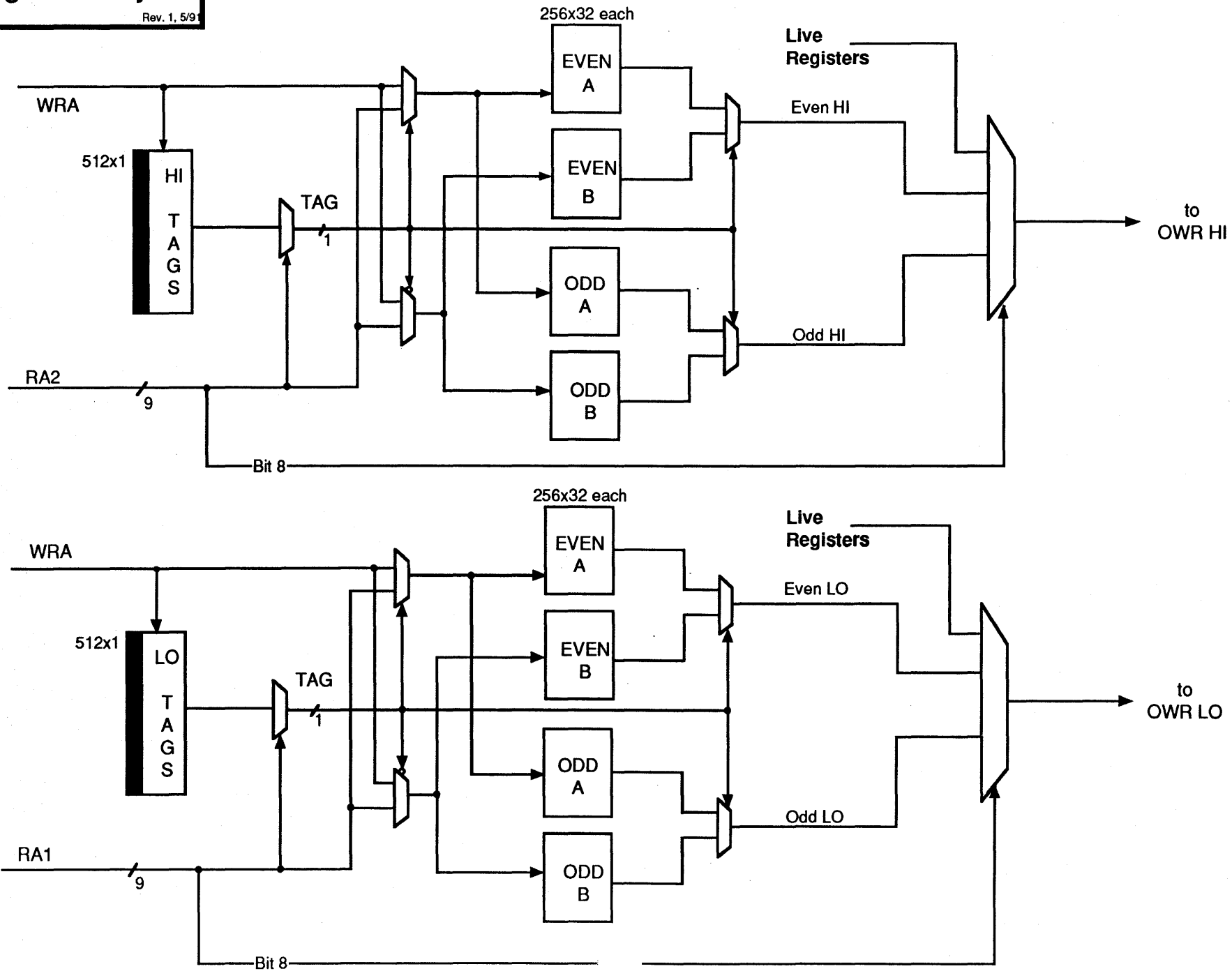
- **Bypasses can buy some of this back.**
  - RR Bypass
    * RR data sent directly into the adder at same time it's written to GPR.
    * Saves 1 cycle over no bypass.
  - OWR Bypass
    * OWR data bypassed into the adder.
    * Only works if bypassing from _____ instructions.
    * Saves 2 cycles over no bypass.
  - EAG Result Bypass
    * EAG complex duplicates ALU calculations being done in E-unit.
    * Done on NR, AR, ALR, LA, SLL, SLA (shift amounts of 0, 2, or 3).
    * Saves up to 5 cycles over no bypass.

EGI Timings

```
GPR modifying inst.  D A T B X W
W/O Bypass                     D A T B X W
RR Bypass                    D A T B X W
OWR Bypass                 D A T B X W
EAG Result Bypass      D A T B X W
```

Register Array

Rev. 1, 5/91



Register Array

Register Array - per Sequoia architecture

- Includes all architectural registers (GPRs, Control Registers, Timer Registers, etc.) except Floating Point and Vector Registers.
  - Superset of IBM defined registers.
- Defined to be a 256x32 bit array with each register in a defined location.
  - IBM defines as a variety of registers. Sequoia consolidates them all into one entity.

Register Array - as implemented

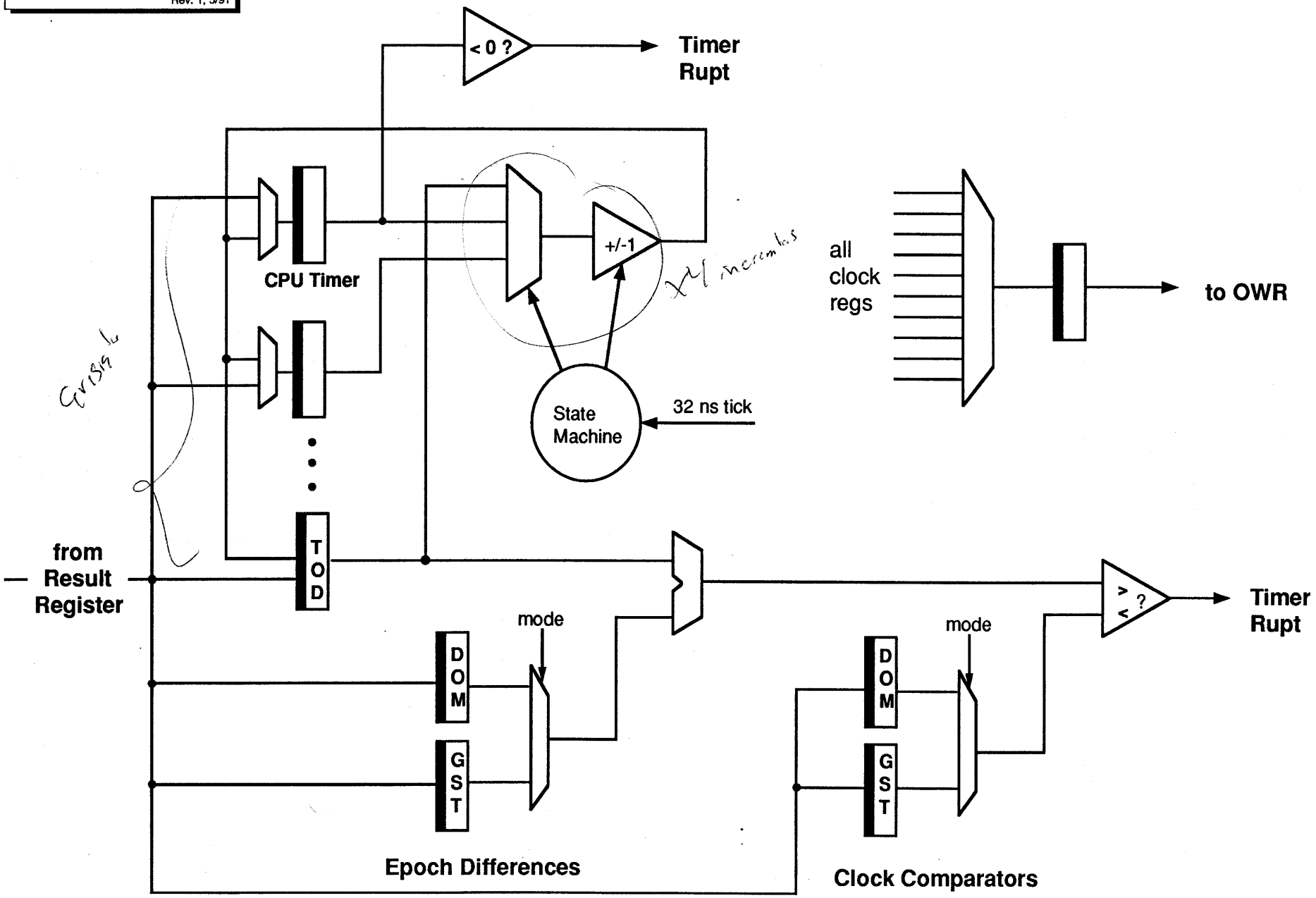
- 512x32 array implemented in RAM (256 scratch registers provided for µcode use).
- All architectural registers implemented in 1 of 3 types:
  - RAM Register: the only copy is in the RAM array.
  - LSI copies: the register is in RAM, but LSI copies are kept elsewhere.
  - Live registers: the only copy that's accurately maintained is in LSI. The RAM location is reserved but not used.
- RAM array capabilities (requirements) include:
  - concurrent read and a write to _____.
  - can write even and odd registers in parallel for _____.
  - read any two registers in parallel for _____.

Implementation scheme

- Concurrent Read/Write:
  - 2 banks of RAM (A and B) implemented, each large enough to hold all the registers.
  - For a given location, only 1 bank contains the current, up to date copy.
  - A 1 bit LSI TAG, one per location, indicates which bank is up to date for that location.
- Writing register pair:
  - Even and Odd GPRs are put in separate RAMs.
- Reading 2 registers:
  - Duplicate this whole scheme to provide a second read port.

Timer Complex

Rev. 1, 5/91



Timer Complex

Time Of Day clock

- Always running. Keeps track of "absolute" time.
- Architecture requires several versions.
  * Current Domain
  * Macrocode
  * Guest running in current domain
- A Macrocode TOD is maintained in a register. Epoch Differences provide the offsets for the other versions.

Comparators

- "Alarm clocks". The TOD is compared with these values and a rupt is generated when the TOD exceeds them.
- Two versions, Domain and Guest.

CPU Timer

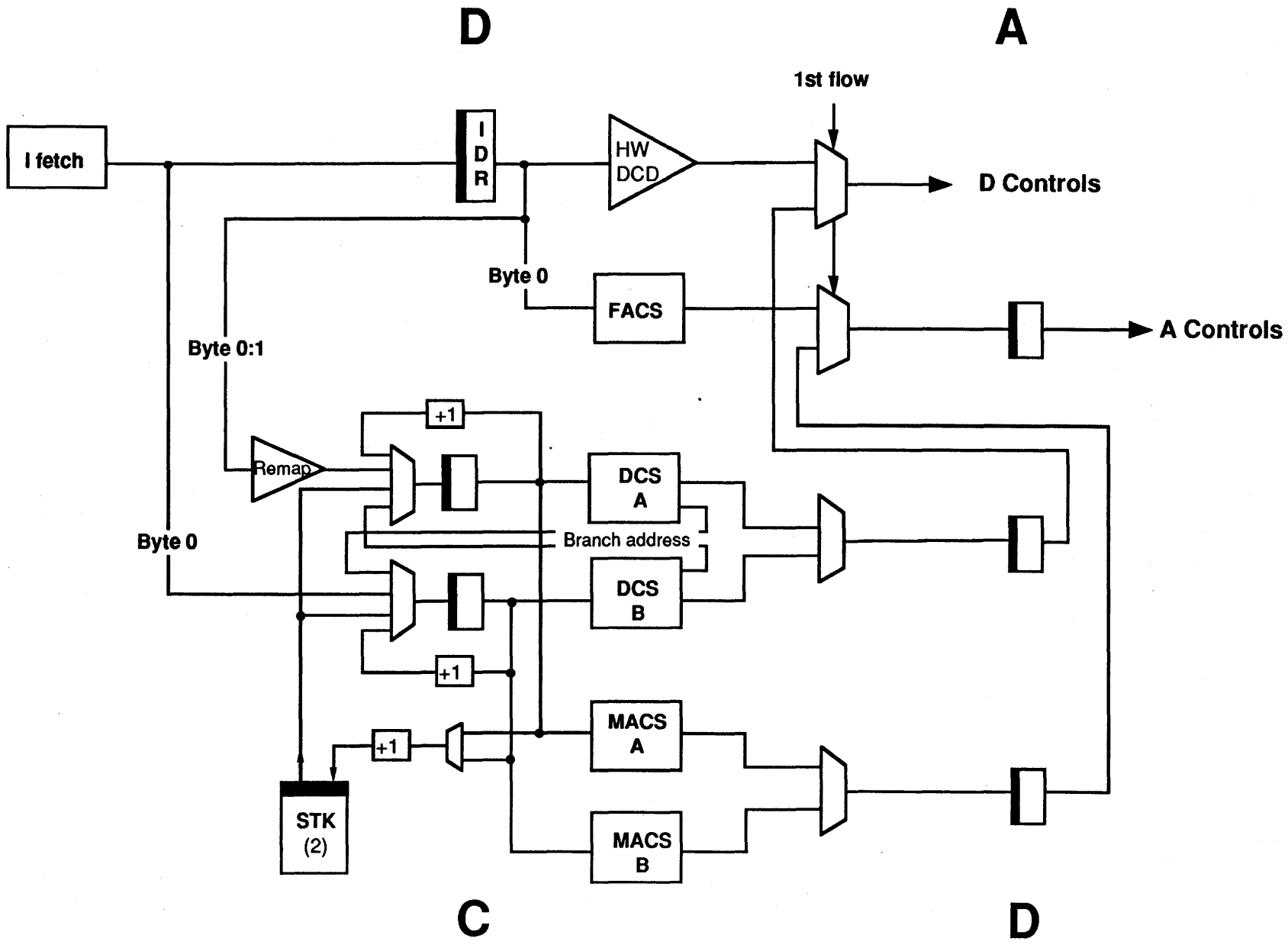
- Only counts CPU time (i.e. when CPU is executing).
- Counts down to zero, then sends a rupt. Useful for time slicing.

Implementation

- Registers hold the current value of the various timers.
- They're multiplexed through an incrementor/decrementor to get updated once per 32 ns.
- 32 ns tick from oscillator card provides timing.
- The TOD is adjusted for Epoch offset, then compared with Comparator register to see if a rupt is needed.
- All registers loaded from the RR, and can load the OWR (via the _____).

I-unit Control Store

Rev. 1, 5/91



Control Store

First flow of an instruction algorithm

- D-cycle control points hard are wired.
- A-cycle and subsequent control points come from FACS (First A-cycle Control Store).
  - 256x96 RAM (including parity).
  - Addressed by byte0 (opcode byte) of IDR.

Second and subsequent flows

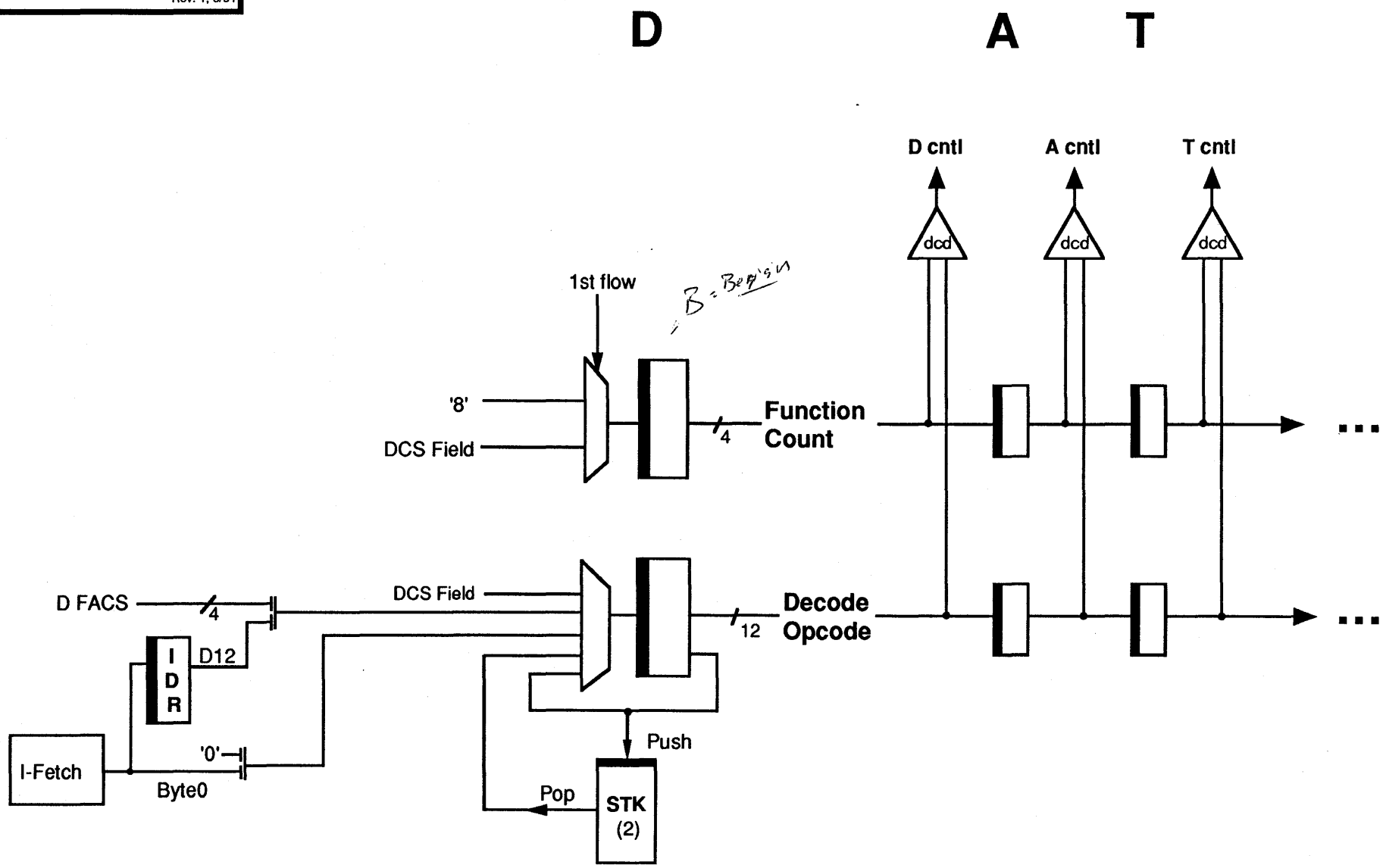
- D-cycle control points come from DCS.
  - Two, 1Kx80 (including parity) banks.
  - Branches always go to the other bank. No branch penalty.
  - 2 deep _____
  - Starting address is _____.
    - * Always starts in bank B.
  - For 2-byte opcodes, the opcode is remapped into 10 bits for the CSADR.
    - * _____ flow is the first unique control store access for a 2-byte opcode.
    - * Always in bank A.

- A-cycle and subsequent control points come from MACS.
  - Same address structure as DCS, just different RAMs and control points.

Decode Opcode

Rev. 1, 5/91

TSR



Decode Opcode

- **Decode Opcode goes down the pipe with each flow.**
  - Hardwired control points are generated by decoding the Decode Opcode.
  - Sparse, stable control points are candidates for hardwiring.

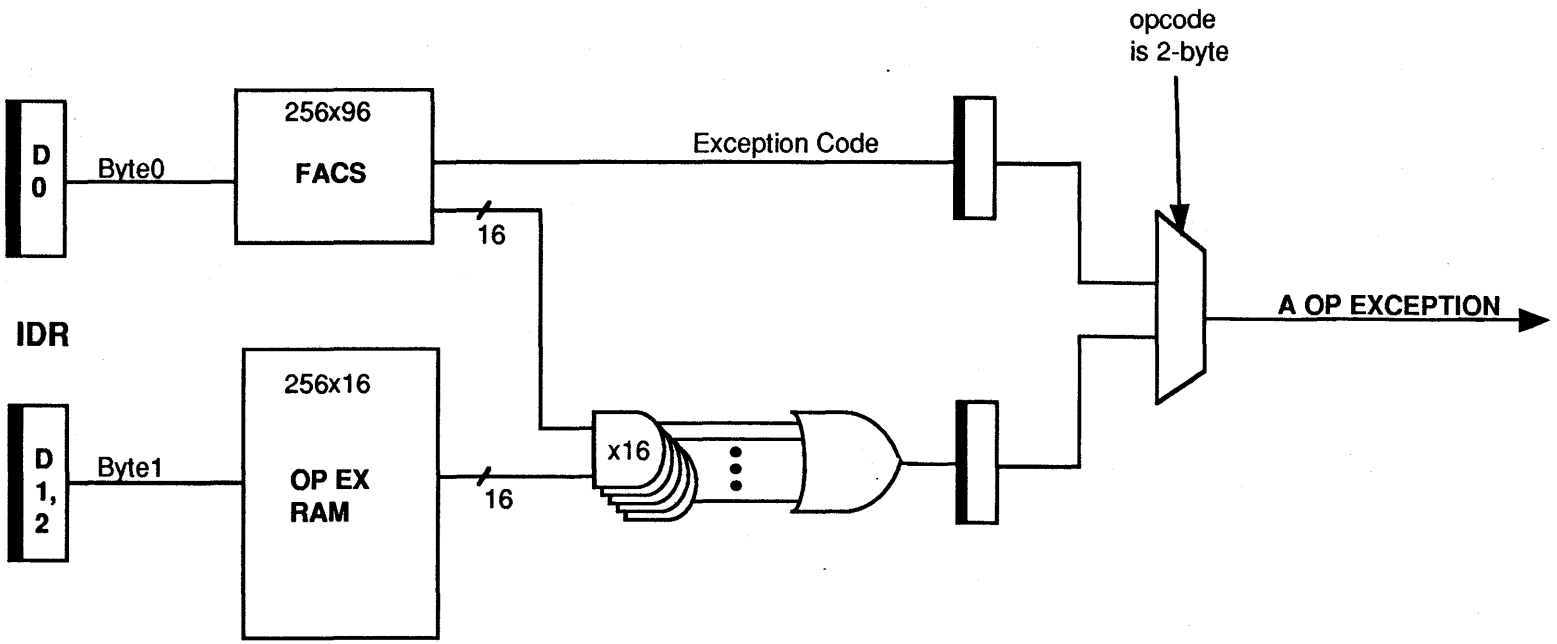
- **The Decode Opcode has two fields:**

1. Decode Opcode
   - 8 bits.
   - original source is _____.
   - Other sources include:
     * Value from previous flow. Multi-flow algs usually keep the same value throughout.
     * D1:2 + 4 bit D-cycle FACS field for _____. Unique DCD OPCD on _____.
     * 2 deep stack for _____.
     * DCS field used to change to a new value when you run out of Function Counts.

2. Function Count
   - Used to differentiate flows in a multiple flow algorithm (all have the same Dcd Opcd).
   - Word "Count" is a misnomer as this field doesn't just increment.
   - Can be reused if different μinstruction flows have same hardwired control points.
   - Starts at 8 on first flow.
   - Sourced from _____ on subsequent flows.

Operation Exceptions

Rev. 1, 5/91



Op Exceptions

- **For 1-byte opcodes, comes from _____.**

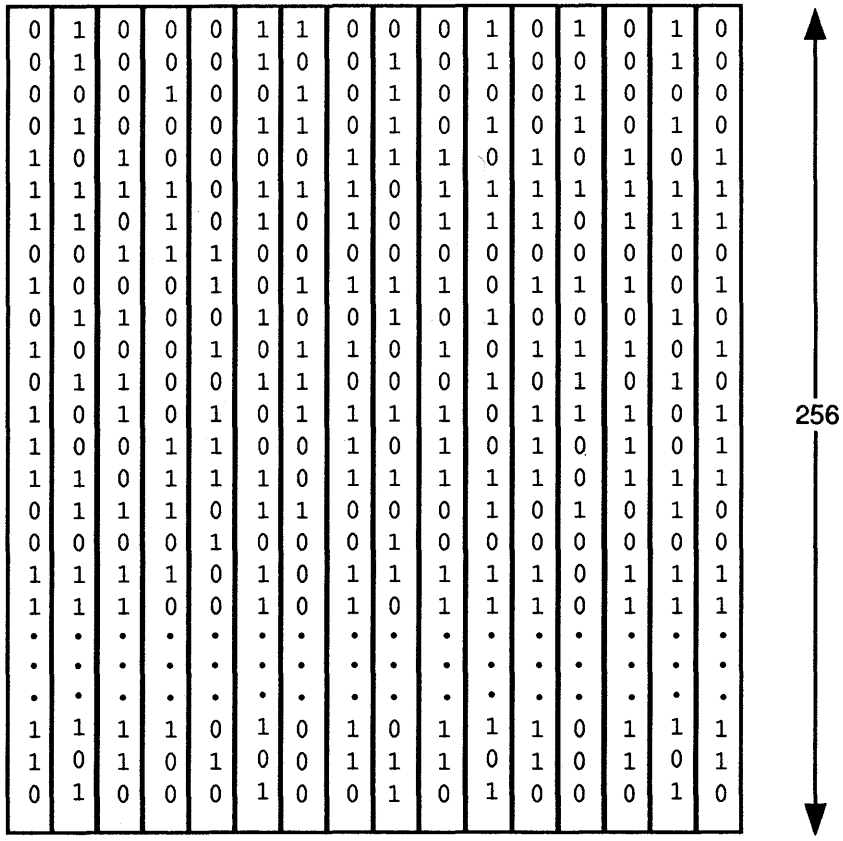
- **For 2-byte opcodes, use OP EX RAM.**
  - 256x16 RAM.
  - Addressed by the second byte of the opcode.
  - Each column belongs to one 2-byte opcode family (same first byte).
  - FACS field selects which column to use.
  - Selected bit indicates opcode validity for that 2-byte opcode.

- **Final selection chooses between the OP EX Ram (for 2-byte opcodes) and the FACS (for 1-byte opcodes).**

TSR

Opcode Families

Byte 1 →



Interlocks

General - all stages

- Inhibit pipe - signal used to freeze pipe state (e.g. for error recovery).
- Pipeline interlock - the downstream stage is valid and is interlocked.

D-cycle Interlocks

- Execute-Generate Interlock - prior instruction is modifying a GPR needed for EAG.
- Programmed Delay Interlock - µcode can force an interlock.
- Overlap Interlock - if overlap turned off, wait for pipe to clear.
- OWR Interlock - OWR EGI bypass shares a bus with RR. If RR is using it, can't bypass.
- I-fetch TLB Validate Interlock - waiting for S-unit access to do an IF TLB validate.
- Domain Interlock - like EGI but for some Sequoia registers.
- D-cycle Control Store Parity Error
- Access Register Interlock - deals with Access Registers, which we're ignoring.

A-cycle Interlocks

- Operand Priority Interlock - waiting for priority into the S-unit.
- A-cycle Control Store Parity Error
- ALB Interlock - deals with ALB, which we're ignoring.
- A eXception Valid Interlock - special interlock to fix a bug.

T-cycle Interlocks

- None.

B-cycle Interlocks

- BALRUS Interlock - BAL and RUS use CC as data. Need to get it set then pass to OWR.

X-cycle Interlocks

- Fetch Data Interlock - waiting for data from S-unit.
- E-Unit Busy Interlock - E-unit busy processing data.
- Condition Code Interlock - waiting for CC setter to allow branch decision.
- Syscom Interlock

W-cycle Interlocks

- None.

- This page intentionally left blank -

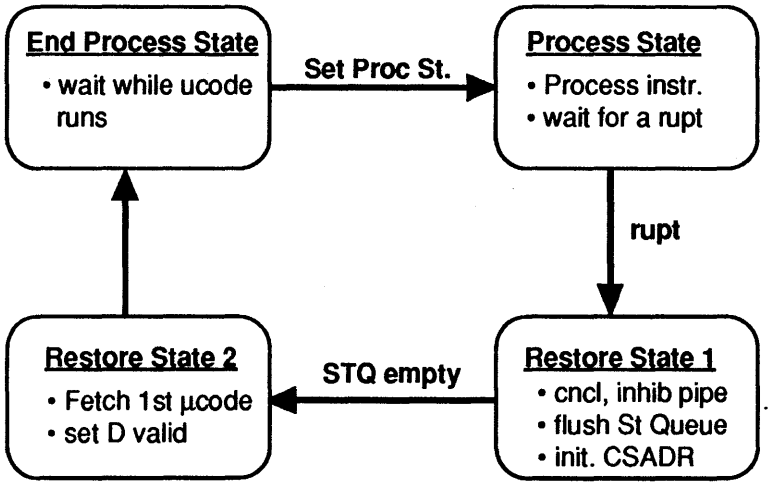
Process switching per the POO

- 6 classes of interrupts:
  - External: Timer rupts, plus some miscellaneous rupts.
  - Program: detected during instruction execution.
    * e.g. overflow, translation exception, operation exception (illegal opcode)
  - Machine check
  - Supervisor Call: this is an instruction.
  - I/O: initiated by conditions or events in the I/O subsystem.
  - Restart: used by console or another CPU.

- Upon taking the rupt:
  1. Stop processing the current instruction stream.
  2. Store the current PSW as Old PSW into a fixed location in page 0.
  3. Store an interrupt code describing the interrupt (into a fixed location).
  4. Load the New PSW from a fixed location and start processing.

Process Control State Machine - Normal Process Switch

Rev. 1, 5/91



Process Control (cont.)

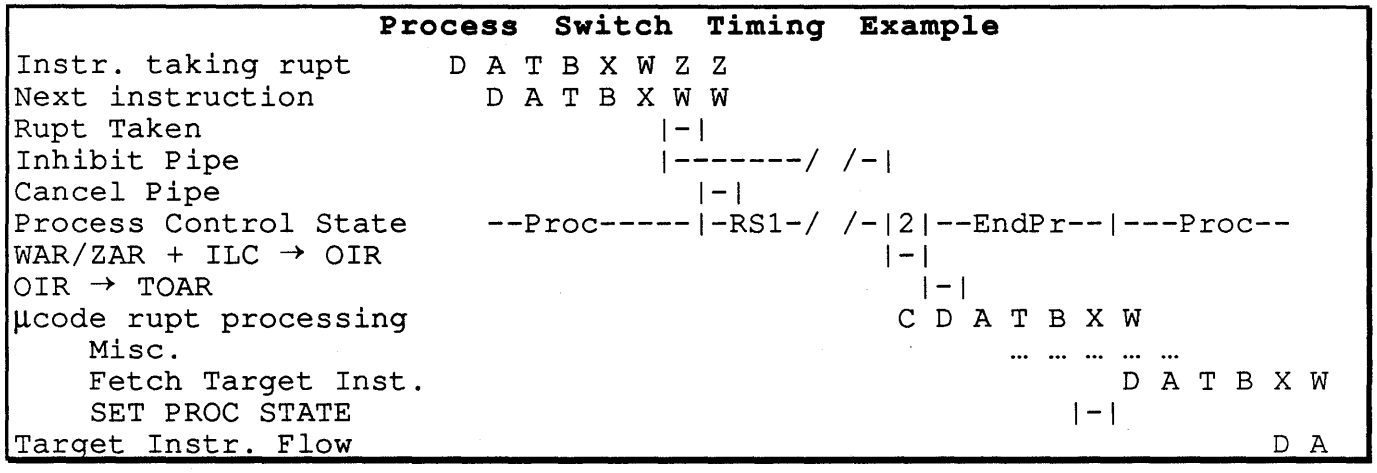
Process Switch implementation

Process Control State Machine:

- Process state - normal state for processing instructions.
  - RS1
  - Cancels/inhibits the pipe and waits for the Store Queue to flush.
  - Starts reconstructing old PSW from ZAR/WAR.
  - Sets up CSAR address (to be loaded next cycle).
- RS2
  - Finishes reconstructing the old PSW (clocks TOAR w/address).
  - Sets up D-valid (to be loaded next cycle).
- End process state
  - waits while µcode does rupt processing.

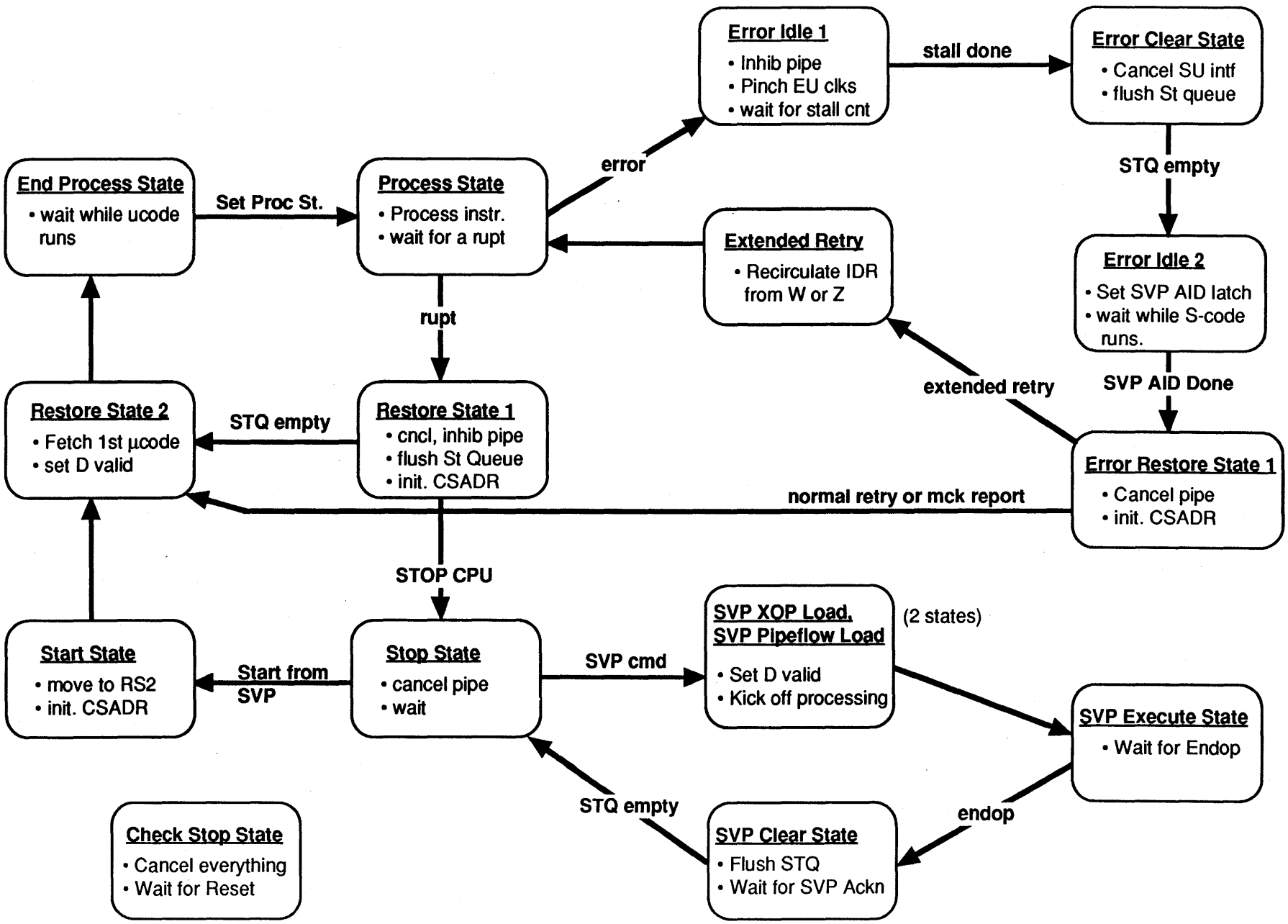
Rupt handling µcode:

- Stores rupt code info, as needed.
- Does any other special handling required (e.g. I/O rupts).
- Loads New PSW. and fetches target instruction.
- Asserts SET PROC STATE, kicking Process Control back into Process State.



Process Control State Machine

Rev. 1, 5/91



Process Control (cont)

- **Start/Stop**

  SVP
  - Issue STOP command (or after a reset).

  State Machine
  - Cancel pipe and wait for Start from SVP. *(Stop State)*

  SVP
  - Issue START command.

  State Machine
  - Set up CSADR and go to RS2. *(Start State)*
  - From there on it looks like normal rupt handling.

- **SVP Op loop**

  SVP *(Stop State)*
  - Scan in an instruction or pipeflow w/clocks off.
  - Turn clocks on.

  State Machine
  - Turn on D valid and start execution. *(Load State)*
  - Wait until done, then flush Store queue. *(SVP Execute, Clear States)*
  - Return to STOP State. *(SVP Clear State)*

- **Error handling loop**

  State Machine
  - Freeze pipe state and E-unit. *(Error Idle 1)*
  - Wait for clocks to go off using stall counter. *(Error Idle 1)*

  SVP
  - S-code repairs damage, then turns clocks back on.

  State Machine
  - Flush Store Queue. *(Error Clear State)*
  - Request SVP Aid, if needed. *(Error Idle 2)*

  SVP
  - With I/E clocks off, S-code assembles MCIC for Macrocode, based on log analysis.

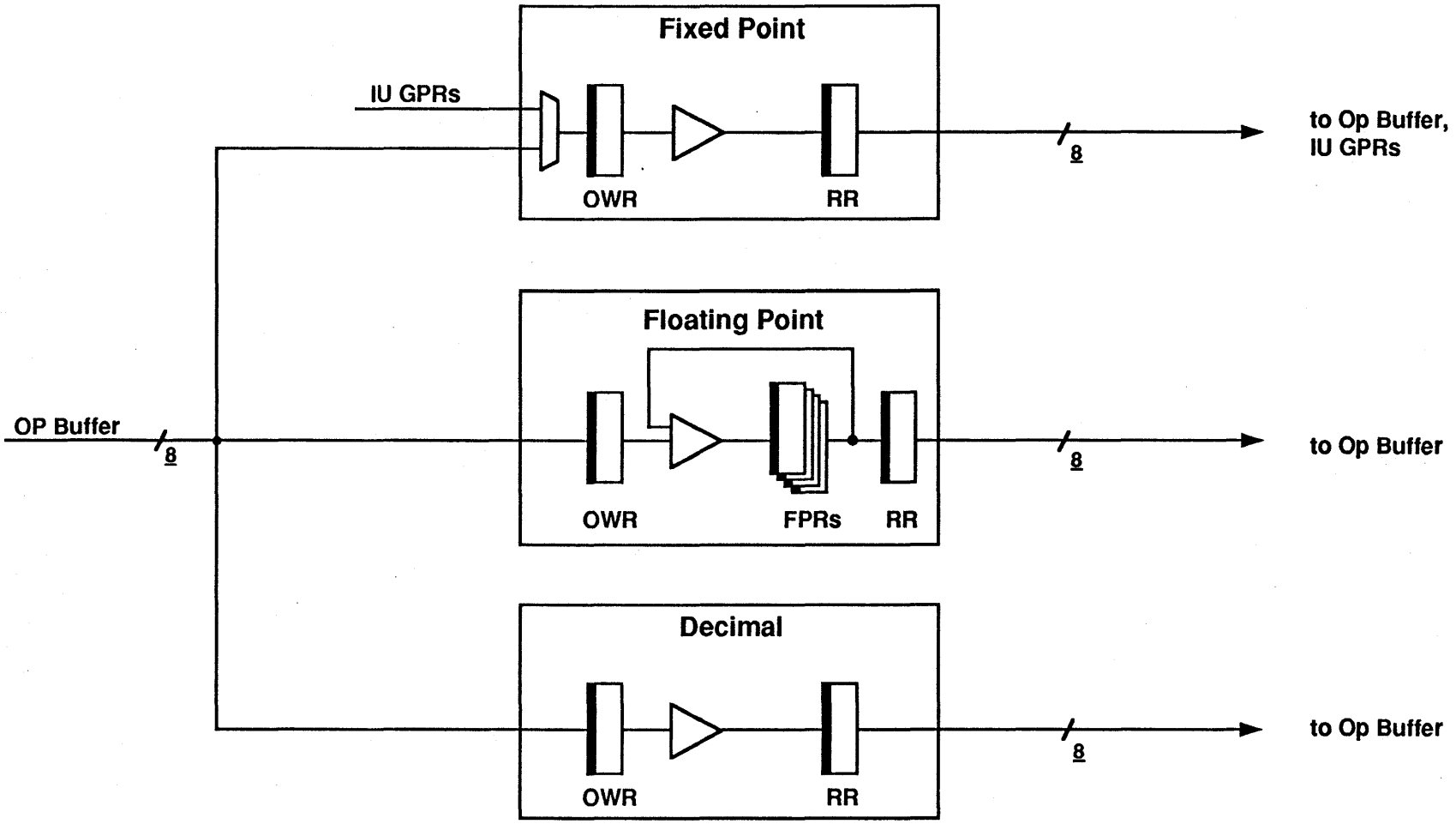
  State Machine
  - Reconstruct Old PSW and go to RS2. *(Error RS1)*
  - If not past retry point, refetch from Old PSW. Otherwise, load CSAR to point to µcode to do rupt processing and fetch first µinstruction. *(RS2)*

  µcode *(if no retry)*
  - Store OLD PSW and MCIC.
  - Load New PSW and start processing.

E-unit

E-unit
Basic Blocks

Rev. 1, 5/91



Sub-units

The E-unit is made up of 3 sub-units:

Floating point

- Handles floating point calculations, per the Floating Point chapter in the POO.

Decimal

- Handles decimal calculations, per the Decimal chapter in the POO.

Fixed point

- Handles everything else, especially the General Instruction chapter in the POO.

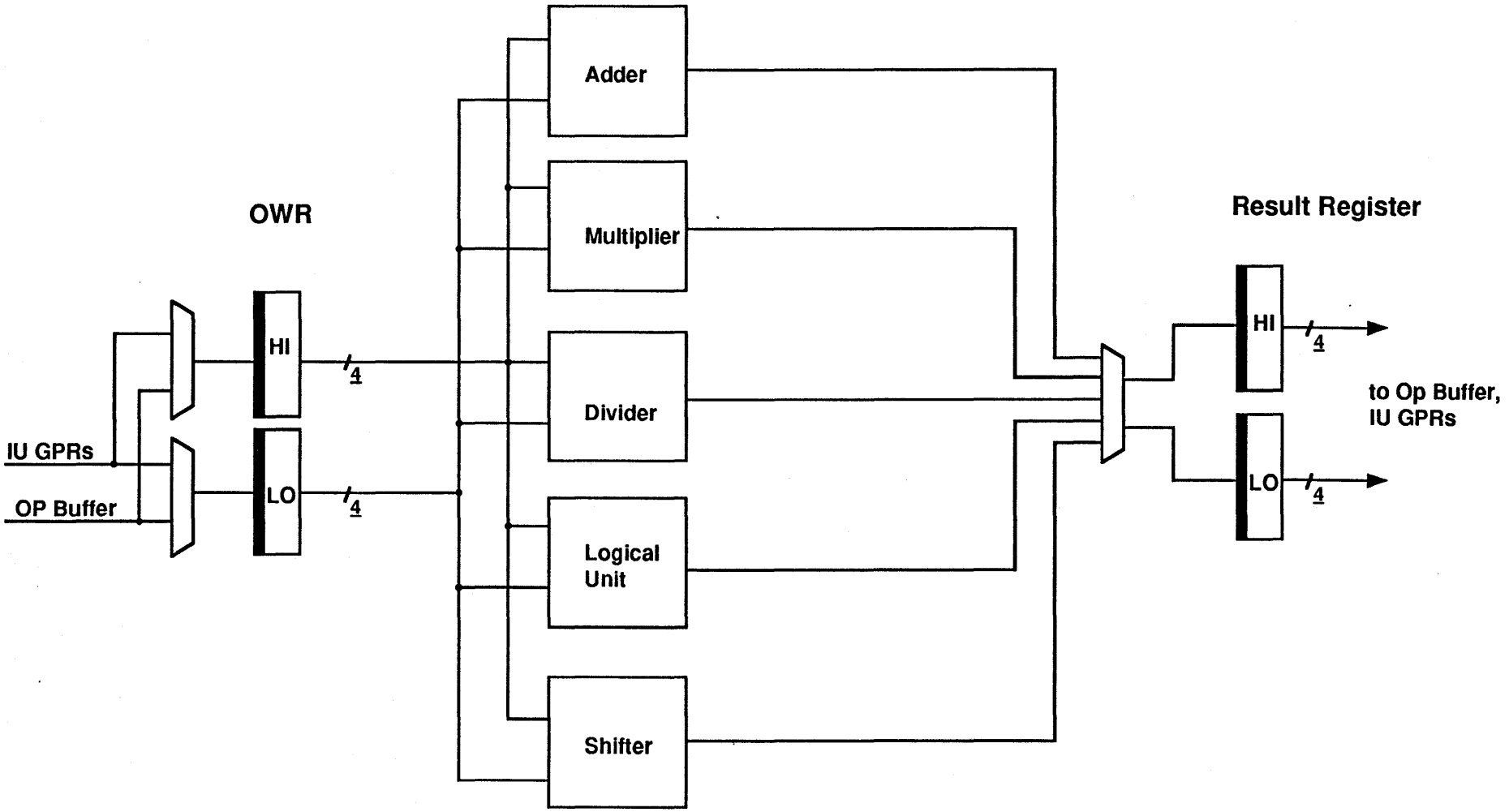
Each sub-unit has its own versions of the OWR and RR.

- This page intentionally left blank -

Fixed Point

Fixed Point Basic Blocks

Rev. 1, 5/91



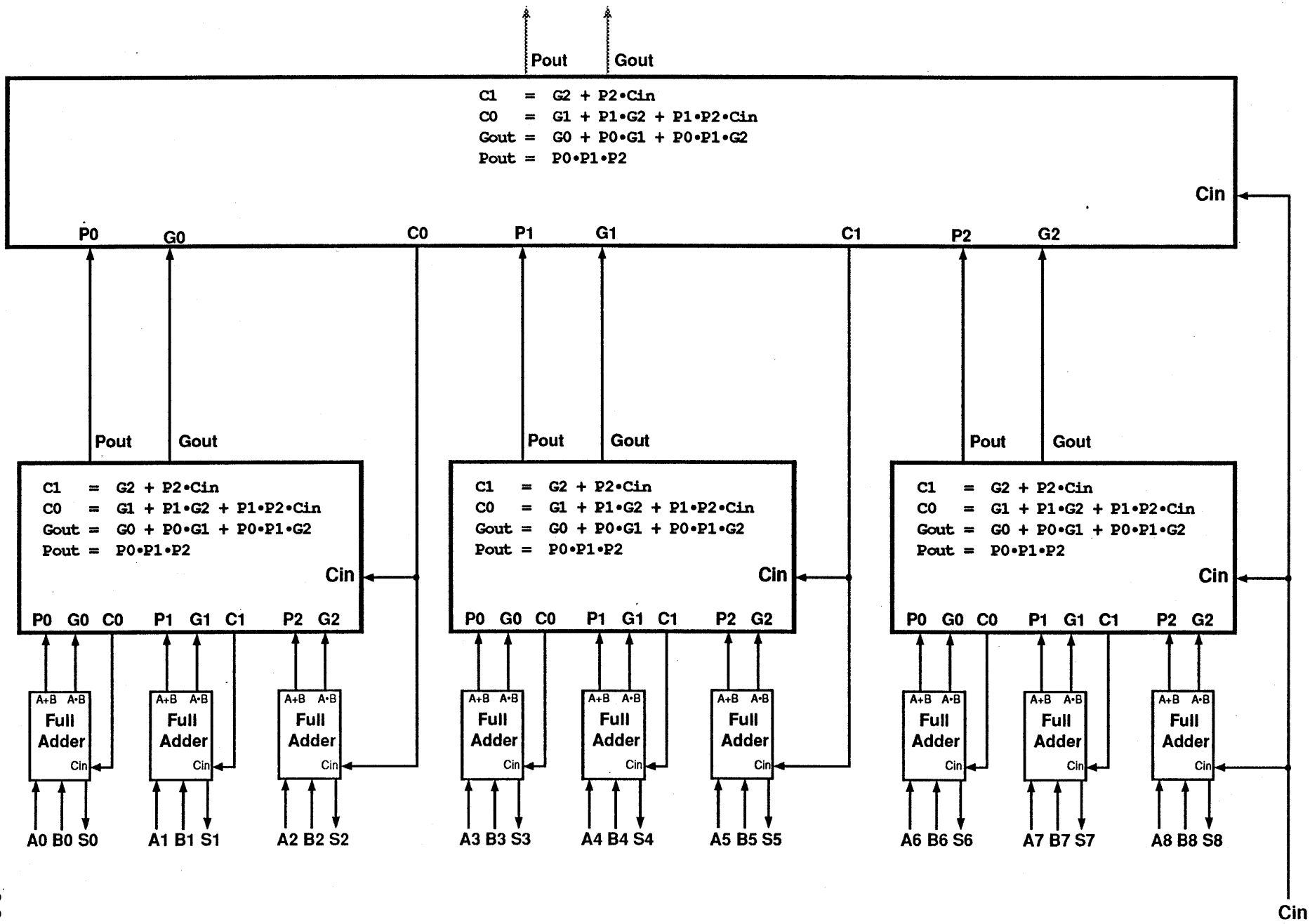
Fixed Point - Basic Blocks

- **The fixed point contains 5 fairly independent blocks:**
  - Adder/subtractor
  - Multiplier
  - Divider
  - Logical Unit
  - Shifter

- **Data is in two's complement notation.**
  - Halfword, word, and doubleword lengths.

9 bit CPA

Rev. 1, 10/91



9-bit Carry Propagate Adder

EXAMPLE ONLY, NOT IN THE DESIGN!

- Used here to illustrate concepts.

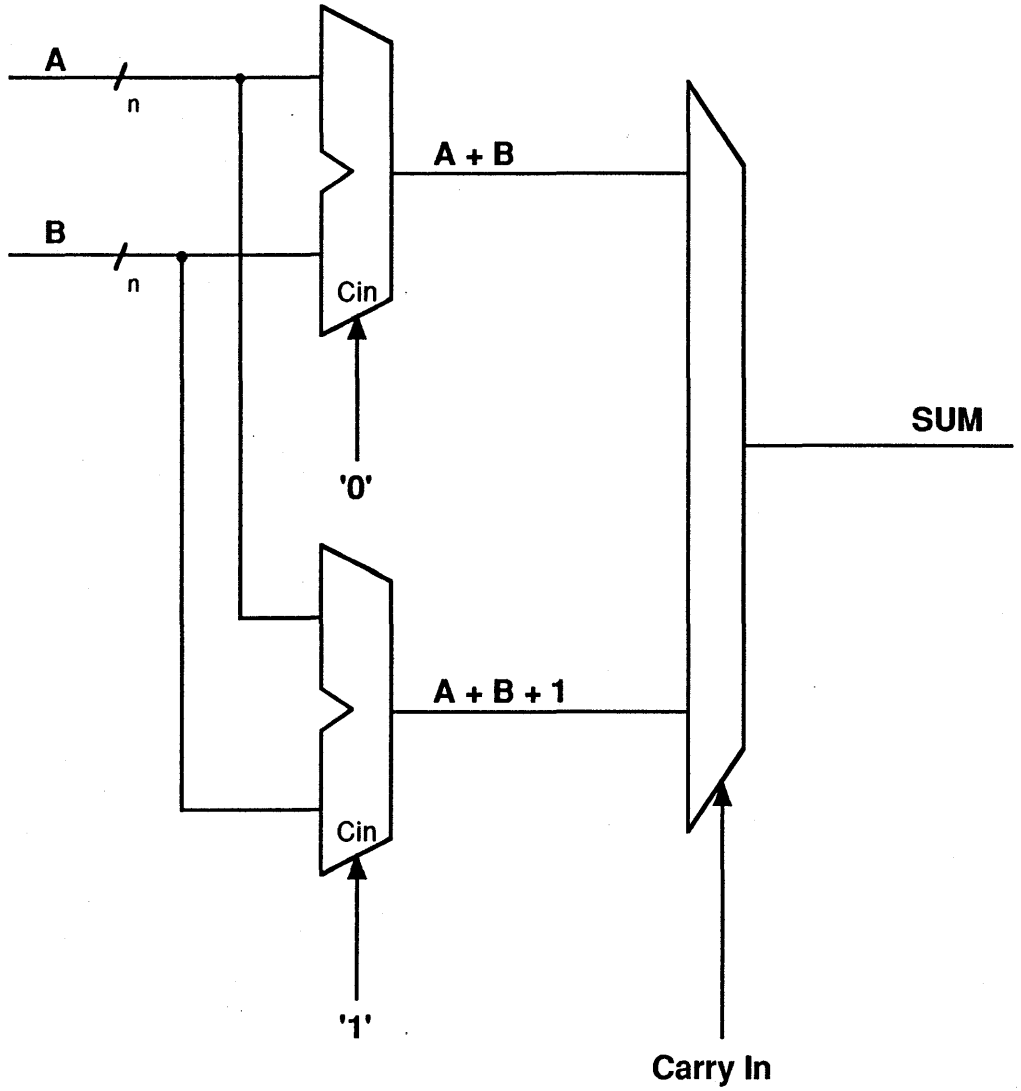
- **Full Adder**
  - 1 per bit.
  - Sums A_n, B_n, and C_{in}.
  - Also calculates Propagate and Generate for each bit.
    * $P_n = A_n + B_n$ (i.e. a carry into this bit will cause a carry out of this bit).
    * $G_n = A_n \cdot B_n$ (i.e. a carry-out is generated, irrespective of the carry in).
    * Note that P and G are independent of C_{in}.

- **Carry Propagate**
  - Bits grouped by three. Each group has a Carry Propagate Element (my name).
  - Based on the P_n and G_n inputs from the 3 Full Adders, plus the C_{in} to the group:
    * Calculates C_{in} into the top 2 bits (the low order bit gets the group C_{in}).
  - Based on the P_n and G_n inputs only (*not* on C_{in} to the group):
    * Calculates P and G for the 3 bits as a whole.

      P means the 3-bit group will propagate a carry coming in.
      G means the 3-bit group generates a carry-out on its own.
  - Can stack elements to make larger adders:
    * One 3-bit element groups together three lower 3-bit elements to form a 9-bit adder.
    * Elements don't have to be same size at each level. Could make a 12-bit adder with a 4 "bit" element at the highest level.

Conditional Sum Adder

Rev. 1, 5/91



n = arbitrary number of bits

Fixed Point CPA

Conditional Sum Adder

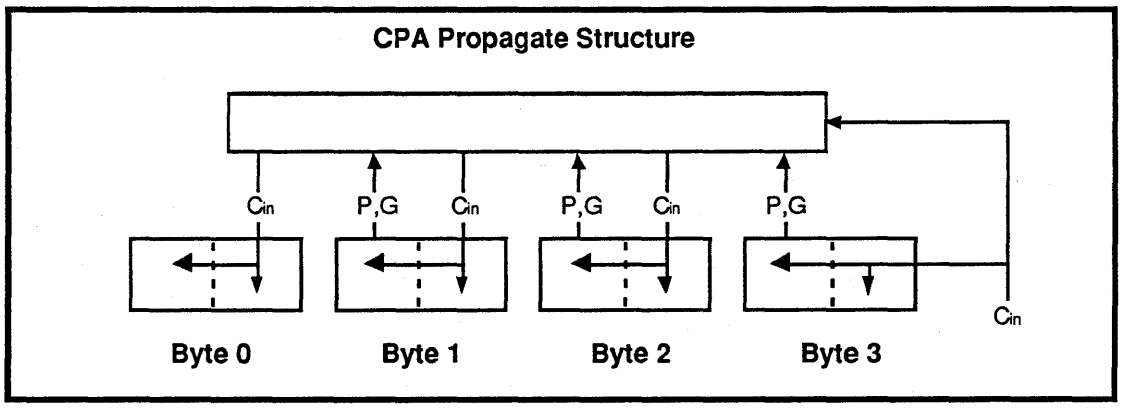
- For a group of bits, calculate the sum with and without the carry-in.
- Let the carry-in select the appropriate sum.

Fixed Point CPA

- **Propagate Structure**
  - 1st level groups bits by 8 to form byte P and G.
  - 2nd level bundles the 4 bytes to form byte C_{in}'s.

- **Carry in structure**
  - Conditional sum done at nibble level.
  - Byte C_{in}'s combined with bit P and G to generate nibble C_{in}'s.
    * Low order nibble gets byte C_{in} directly.
    * High order nibble gets byte C_{in} combined with 4 low order bit P/G's.

- **Adds 32 bits in 1 cycle.**



Multiply Algorithms

16 bits X 16 bits

Standard "Shift and Add" Multiply Algorithm

B0	B1	B2	B3	B4	B5	B6	B7	B 8	B9	B10	B11	B12	B13	B14	B15	times
															1	A
														1		$A \cdot 2^1$
													1			$A \cdot 2^2$
												1				$A \cdot 2^3$
											1					$A \cdot 2^4$
										1						$A \cdot 2^5$
									1							$A \cdot 2^6$
								1								$A \cdot 2^7$
							1									$A \cdot 2^8$
						1										$A \cdot 2^9$
					1											$A \cdot 2^{10}$
				1												$A \cdot 2^{11}$
			1													$A \cdot 2^{12}$
		1														$A \cdot 2^{13}$
	1															$A \cdot 2^{14}$
1																$A \cdot 2^{15}$

Modified Booth's Multiply Algorithm

B0	B1	B2	B3	B4	B5	B6	B7	B 8	B9	B10	B11	B12	B13	B14	B15	times
															-1	A
													-2	1	1	$A \cdot 2^1$
											-2	1	1			$A \cdot 2^3$
									-2	1	1					$A \cdot 2^5$
							-2	1	1							$A \cdot 2^7$
					-2	1	1									$A \cdot 2^9$
			-2	1	1											$A \cdot 2^{11}$
	-2	1	1													$A \cdot 2^{13}$
1	1															$A \cdot 2^{15}$

Fixed Point Multiply Algorithm

Standard "Shift and Add"

- As you traverse Multiplier from right to left:
  - Add in Multiplicand, if Multiplier bit is a 1.
  - Shift Multiplicand left 1 bit (i.e. multiply by 2).
  - Move on to next Multiplier bit.

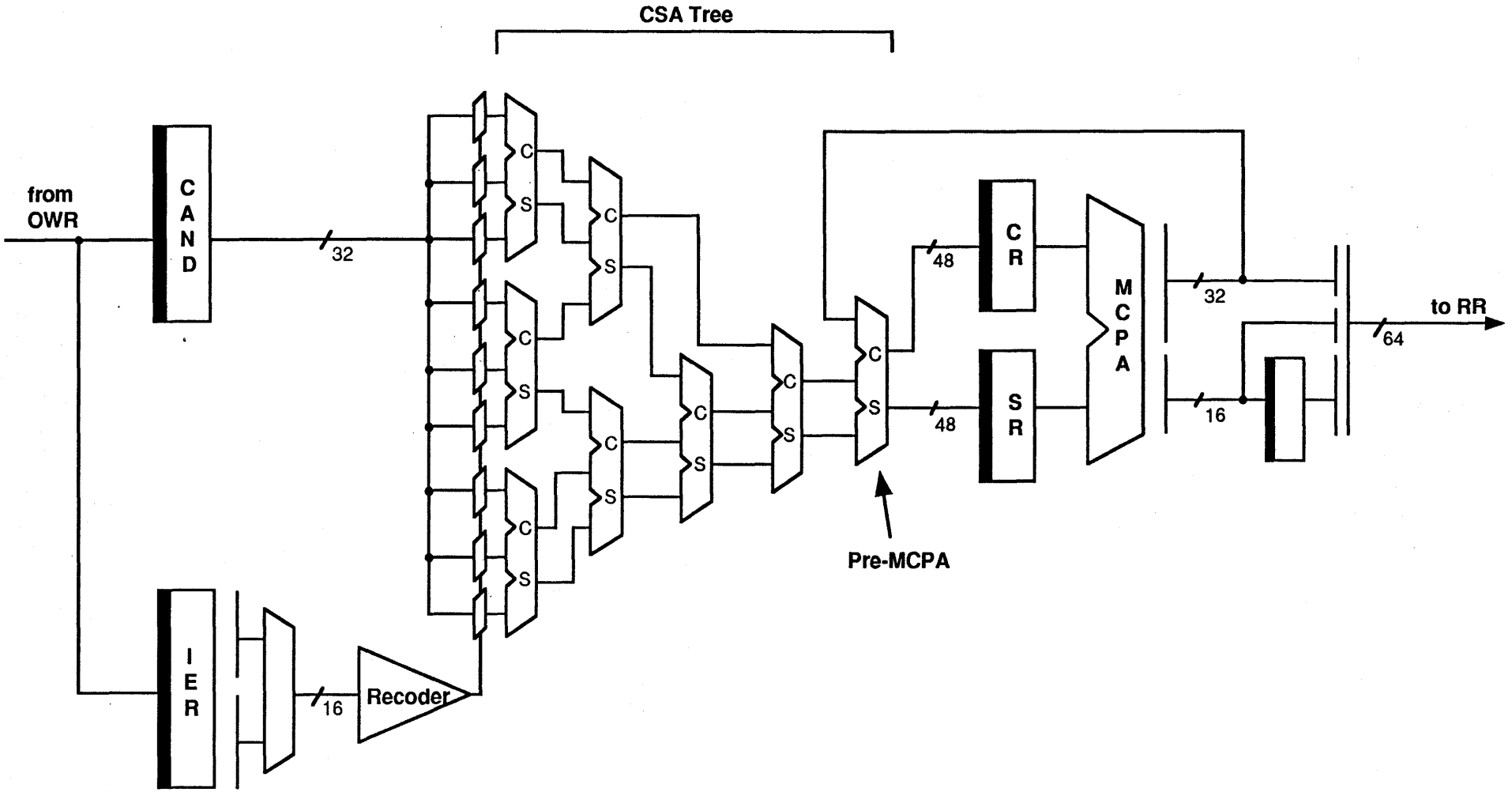
Modified Booth's Algorithm

- Examine Multiplier bits in triplets, moving left 2 bits at a time (i.e. edge bits of triplet are shared with adjacent triplets).
- From most to least significant Multiplier bits, contribution is -2, 1, 1. (See table, p. 3-12).
- Possible values for a given row (i.e. given multiplicand shift amount) are _____. All of these can be generated by _____.

Triplet Value	Multiplicand Select
000	0
001	1
010	1
011	2
100	-2
101	-1
110	-1
111	0

Fixed Point Multiplier

Rev. 1, 5/91



Multiplier Implementation

POO Definition

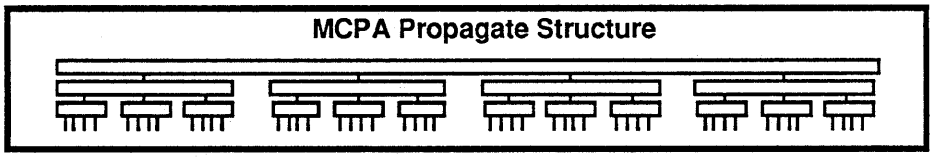
- Multiplies two 32-bit operands to form a 64-bit result to be stored into a register pair.

Implementation

- Breaks 32x32=64 bit multiply down into two 32x16=48 bit multiplies.
- Current MultiplIER half (16 bits) is recoded per Booth's algorithm, then controls shift and adding of MultipliCAND. Generates 9 terms.
- Carry Save Adder tree reduces nine 32 bit terms to two 48 bit terms, then adds in upper 32 bits from prior cycle in the last CSA level.
- Multiply Carry Propagate Adder sums the final two terms.
- Low 16 bits from first cycle are concatenated with 48 bits from 2nd cycle to form final 64 bit result.
- Takes ___ X-cycles.

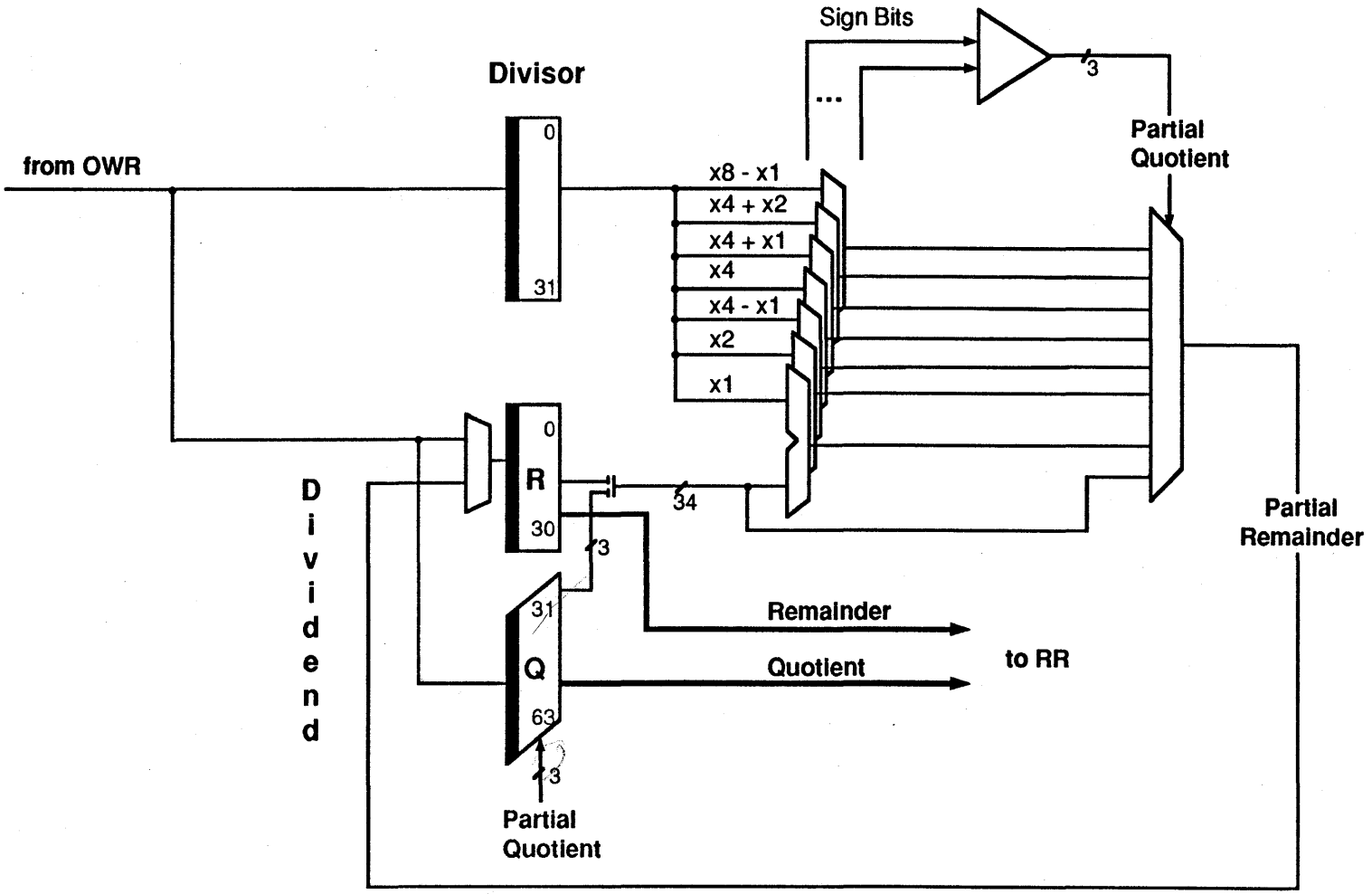
Carry Save Adder

- Technique for multiple operand addition.
- Basic element is a 3 input adder:
  - For each bit position it generates a Sum bit and a Carry bit.
  - Instead of propagating the Carry, it's shifted left 1 bit to form a new operand.
  - Output is two operands, a Sum and a Carry. Thus, the CSA reduces 3 operands to 2.
- By stacking these CSAs into a tree, multiple operands can be reduced to 2, without having to propagate any carries.
- Eventually, a CPA is needed to sum the final two terms. The CPA structure is:



Fixed Point Divider

Rev. 1, 5/91



Fixed Point Divider

POO Requirements

- Dividend - 64 bits
- Divisor - 32 bits
- Quotient - 32 bits
- Remainder - 32 bits

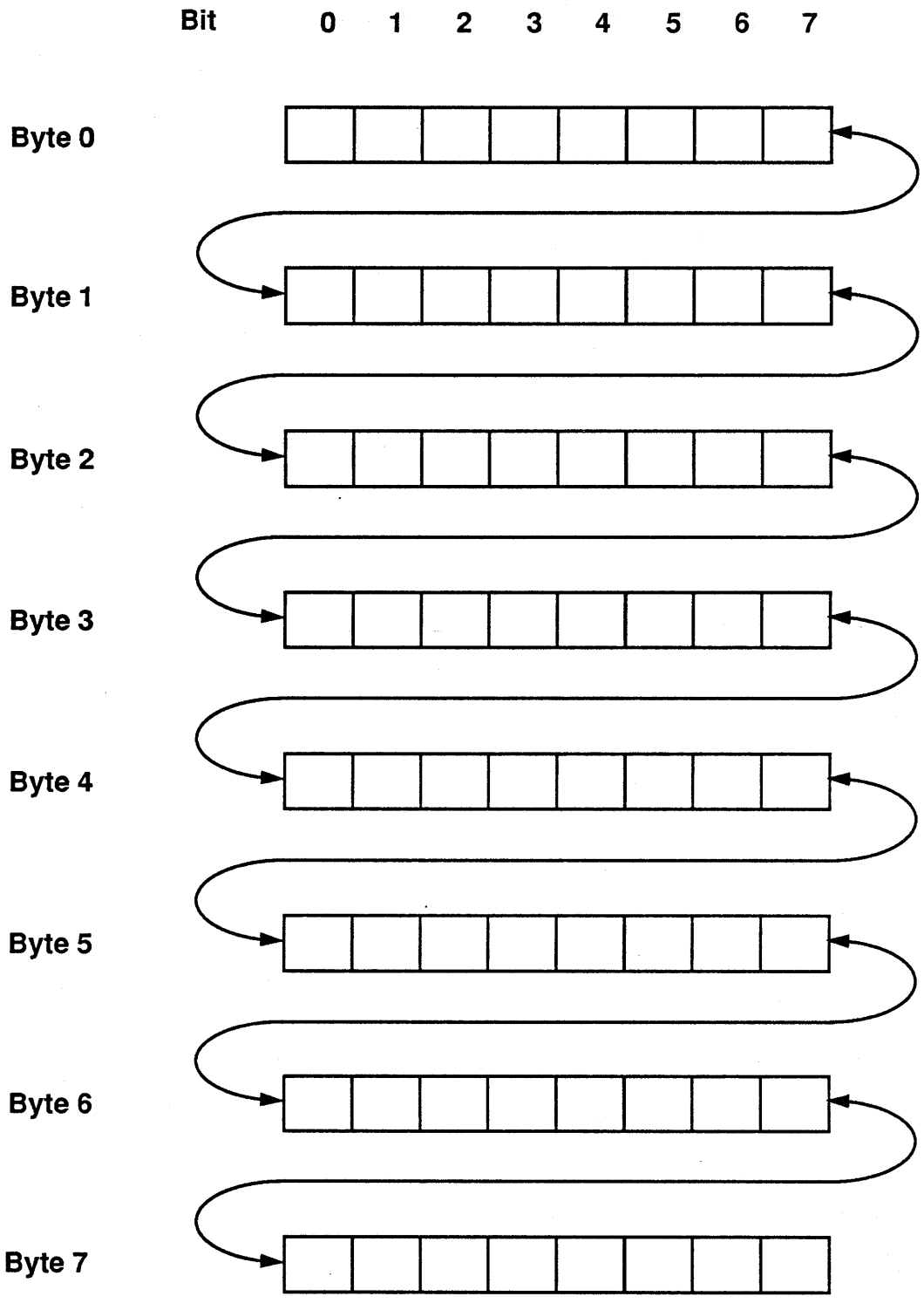
Implementation

1. Load so the Dividend is positive and the Divisor negative. (Paths not shown.)
2. Do trial subtractions (i.e. additions of negative) of 1to7 times Divisor from Dividend 0:33.
   - All 7 trial values are obtainable with shift, complement, and 1 addition. This is built into the adders.
   - Dividend bit 33 is aligned with Divisor bit 31. This allows the Dividend value to be up to 7 times the Divisor value.
3. Select "winning" result back into Dividend 0:30.
4. Left shift appropriate Partial Quotient bits into Dividend 31:63, using the room left by the 3 bits that participated in the subtractions.
5. When done, the Remainder is in the upper half of Dividend, and the Quotient is in the lower half.
6. Does 3 quotient bits/cycle.
7. In the picture the Dividend register is broken into 2 parts, R and Q:
   - R is loaded directly with the selected remainder from the trial subtractions.
   - Q is a shift register; can do 3-bit left shifts.
   - Except for bit 31, R and Q correspond to the Remainder and Quotient at the end.

Note: If the Dividend has excess leading zeros, they can be shifted out (by a multiple of three) via the shifter prior to starting the alg, and the number of quotient iterations is then reduced appropriately. The shift amount is basically _____.

Fixed Point Shifter - Byte Shift

Rev. 1, 5/91



Fixed Point - Shifter

POO Requirements

- Shift double word left or right by 0 to 63 bits.
- Sign bit may or may not be included.
- Shift-in value may be zero or the sign bit.

Implementation

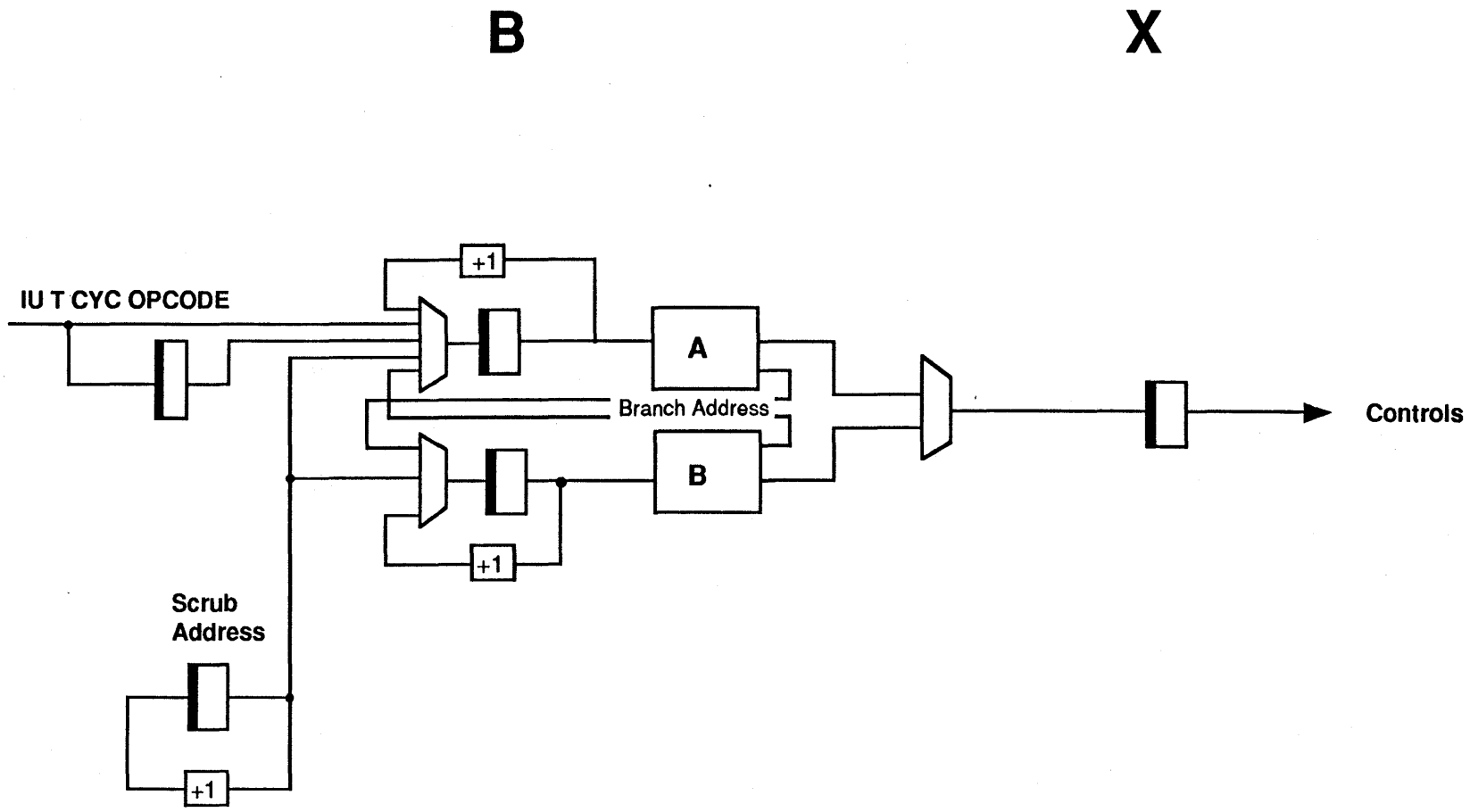
- Shift done in two stages. For shift amount S:
  1. Rotation done within bytes. This rotation is same for all bytes and = _____.
  2. Bits are shifted in byte multiples (maintaining position within byte).

     _____

     _____
- Sign bit and shift-in values are details not covered here.

E-unit Control Store

Rev. 1, 5/91



E-unit Control Store

- **Each Sub-unit has its own Control Store. Basic structure is the same for all.**
  - FX CS is 1024 x 128.x2
  - FP and DU CS are each 256 x 96 x2

- **E-unit µcode is tightly coupled with I-unit µcode.**
  - Especially multi-flow algorithms.
  - e.g. E-unit µcode may assume data will be in the OWR without explicitly checking for it.

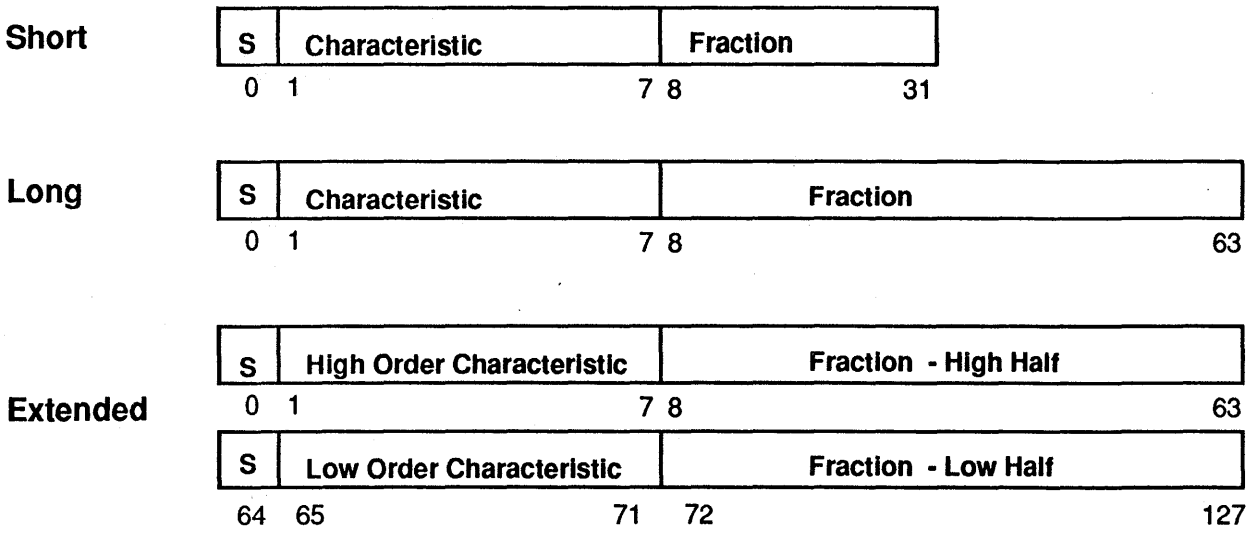
- **The basic control store structure includes:**
  - I-unit sends an opcode in the T-cycle which serves as the starting CS address.
    * Opcode can be held in a register in case _____.
  - Two banks. Increment through 1 bank, branch to the other. Similar to I-unit.
  - Background scrub machine to access CS when sub-unit is idle, searching for errors.

- This page intentionally left blank -

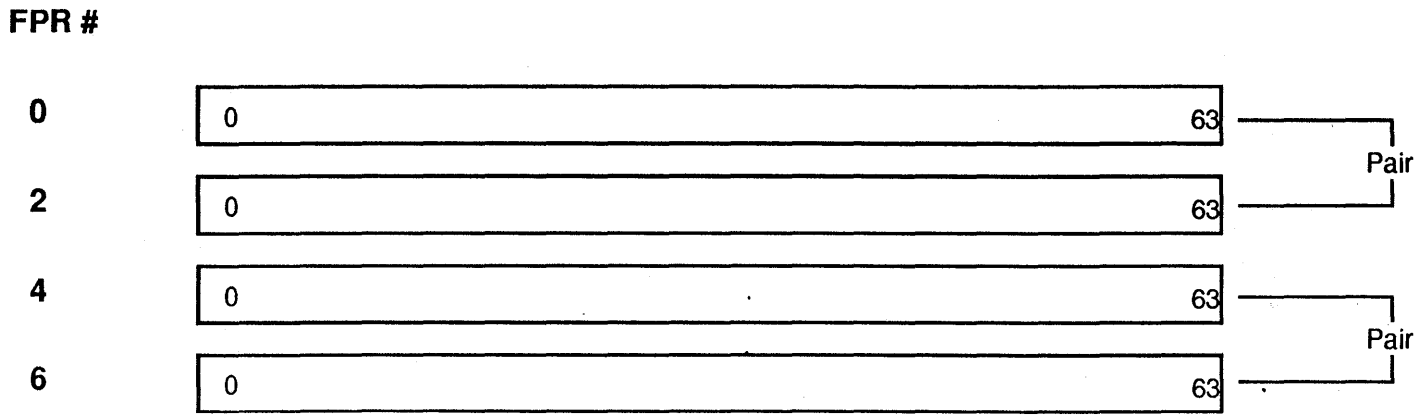
Floating Point

TSR

Data Formats



Floating Point Registers



Floating Point Architectural Elements

Data Formats

- First bit is sign bit.
- Fraction is in Hex with Hex point at the left. (Elsewhere, called Mantissa.)
- Characteristic is exponent (base 16) in excess 64 notation. Thus ...

$$N = \text{Fraction} \times 16^{(\text{Characteristic}-64_{dec})}$$

- Format precision varies (but the characteristic is always the same):

Short:	6 digits	
Long:	14 digits	This is the "standard" format the FPU is built around.
Extended:	28 digits	(note: low order characteristic is ignored during processing, but is set to hi order characteristic - 14 when results are stored)

Floating Point Registers (FPRs)

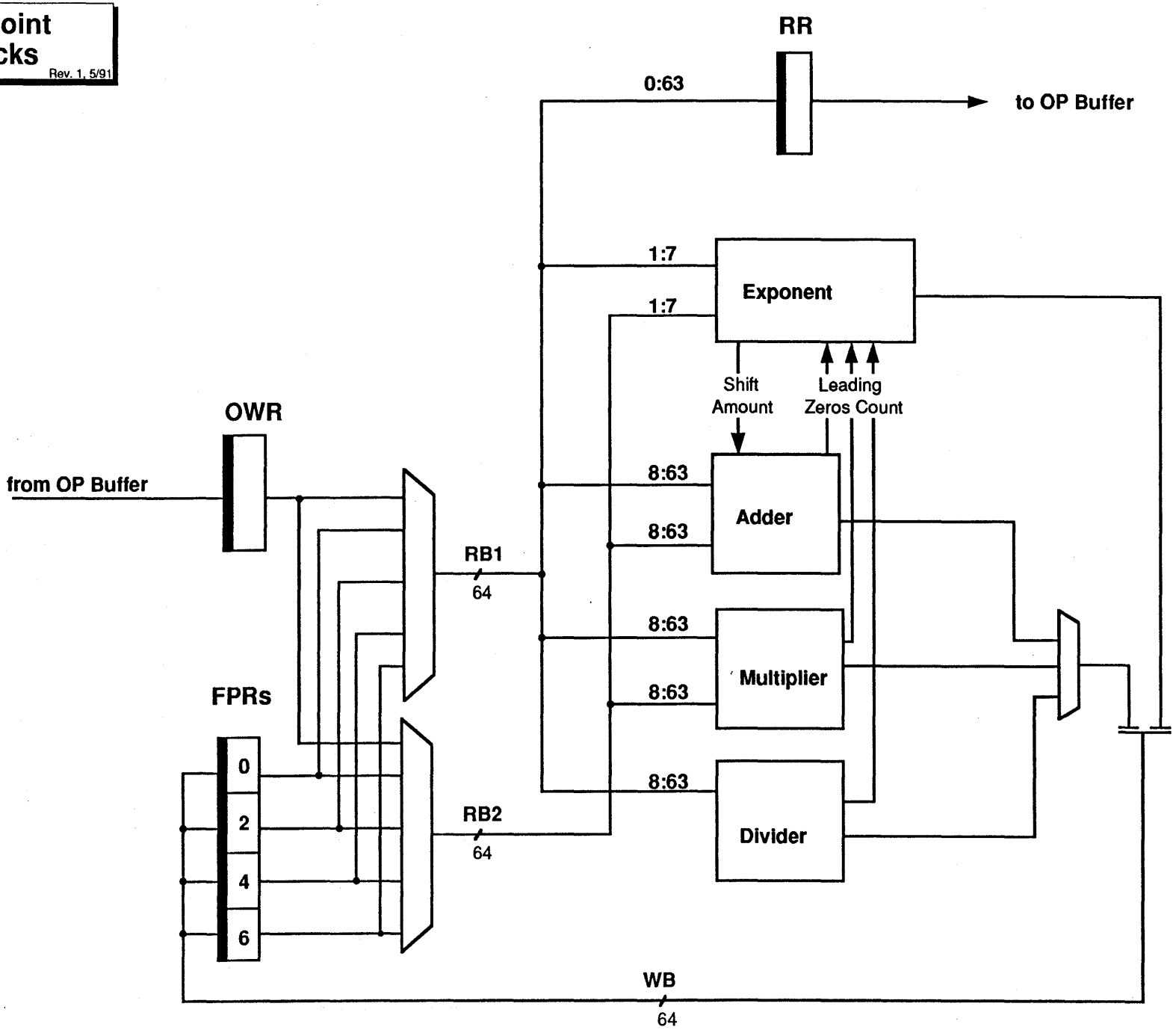
- 64 bits (= long format)
- Register pairs are used for extended operations.
- Only used in Floating Point instructions.
- The Floating Point Unit has the only copy of these registers.

Instruction types

- Add, Subtract, Multiply, Divide, Load, Store
- Some instructions require results to be normalized (leading digit made non-zero).
- See Chapter 9 of the POO for details.

Floating Point Basic Blocks

Rev. 1, 5/91



Floating Point Basic Blocks

- **Buffer data comes into OWR**

- **Two Read Busses select the operand sources.**
  - At least one is an FPR (FP ops are RX or RR).

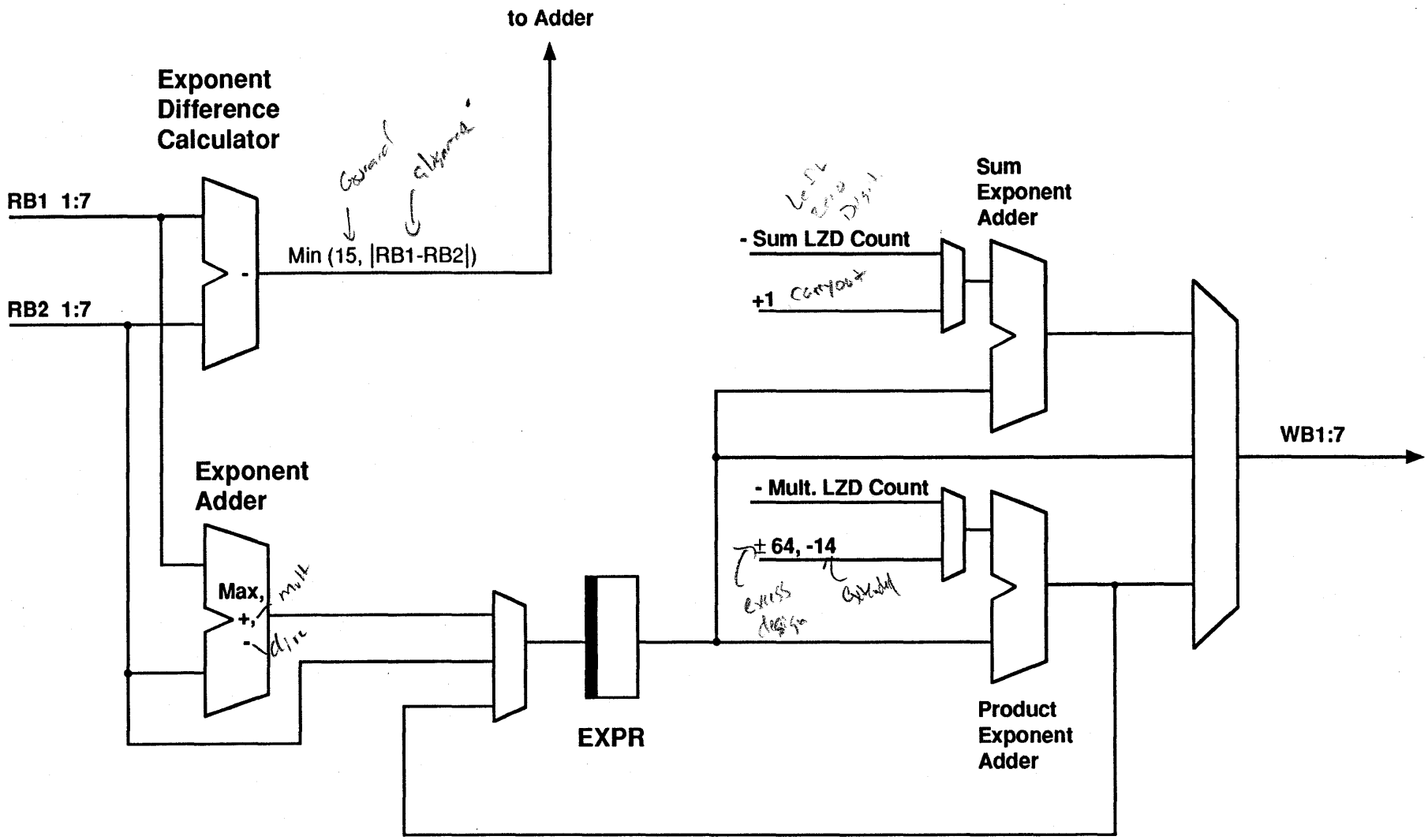
- **Separate sections for fraction addition, multiplication, and division.**

- **One exponent section for all operations**
  - On adds/subtracts, determines alignment shift amount and sends to adder.
  - Receives leading zero digit count for normalization (ie. to decrement exponent).
    (Slight lie in picture - Division LZDC goes through multiplier)
  - Sign bit also handled here, but not included in any pictures or discussion.

- **Write bus writes the results back to the FPRs.**
  - Result Register only sourced from FPRs. Only needed for _____.

Floating Point Exponent Complex

Rev. 1, 5/91



Exponent Complex

Difference Calculator

- Calculates the ABSOLUTE VALUE of the difference between the two exponents and sends to the adder complex.
- The max shift amount for Long Operations is 15. Past that and you're just adding zero.
- An extended shift amount (max is 31) is also calculated and sent out. Not shown.

Exponent Adder

- Calculates the new exponent value and loads into the EXPR.
  - For Multiply/Divide _____
  - For Add/Subtract _____
  - Can also pass through RB2 unchanged, and RB1 has a direct path to EXPR.

Sum Exponent Adder

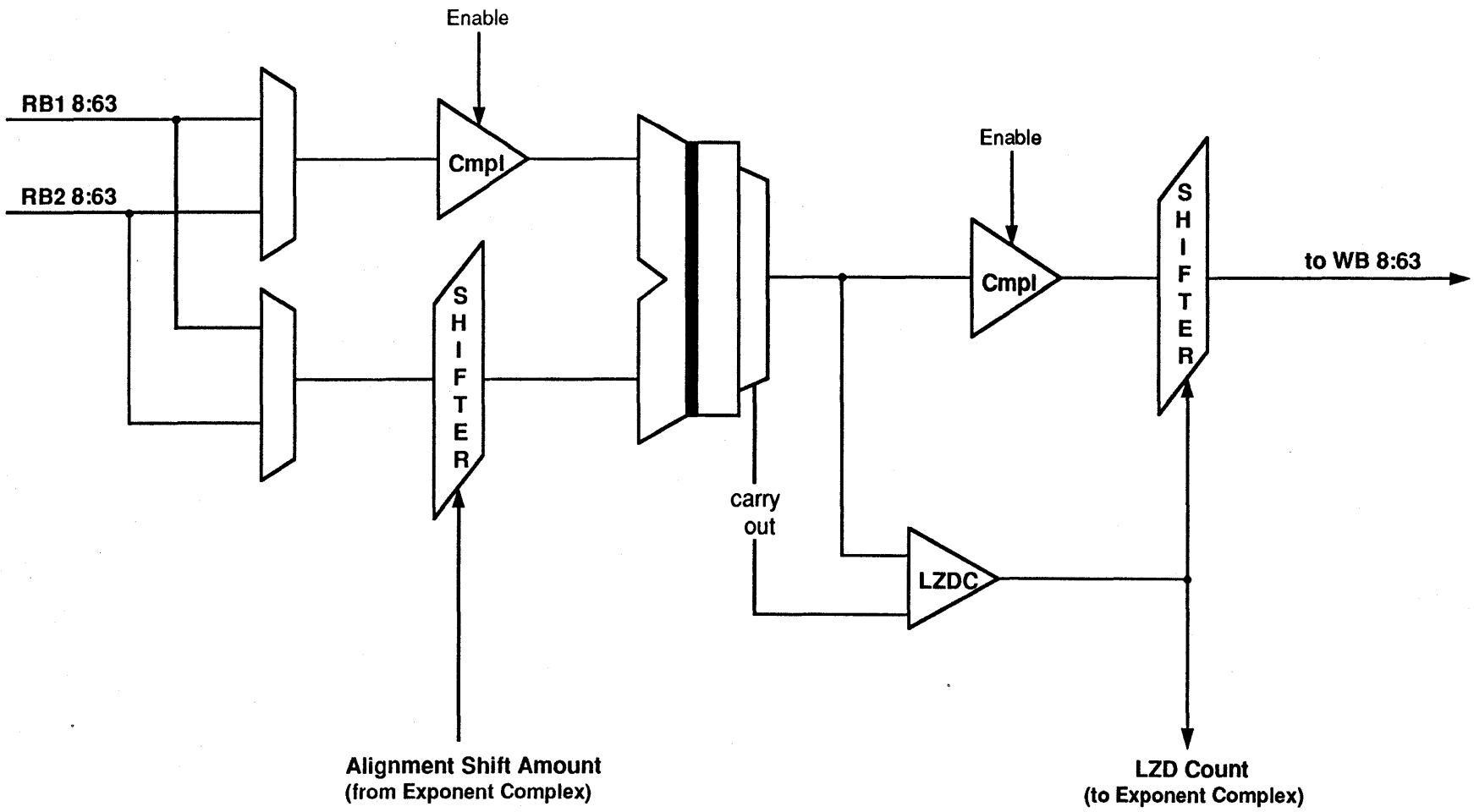
- Decrements exponent by the LZD Count from the adder for normalization.
- Increments on _____
- Sends resulting characteristic out on the Write Bus.

Product Exponent Adder

- Decrements exponent by the LZD Count from the multiplier for normalization.
- On divides, the quotient is accumulated in the multiplier, so the same LZD can be used.
- ±64 input used to _____
- -14 input used to _____

Floating Point Adder

Rev. 1, 5/91



Floating Point Adder

- **Fraction with smaller exponent is right shifted for alignment.**
  - Per the Alignment Shift Amount from the Exponent Complex.

- **The other operand may be complemented.**
  - Complement if _____ of minus signs, where subtraction counts as 1 minus sign.

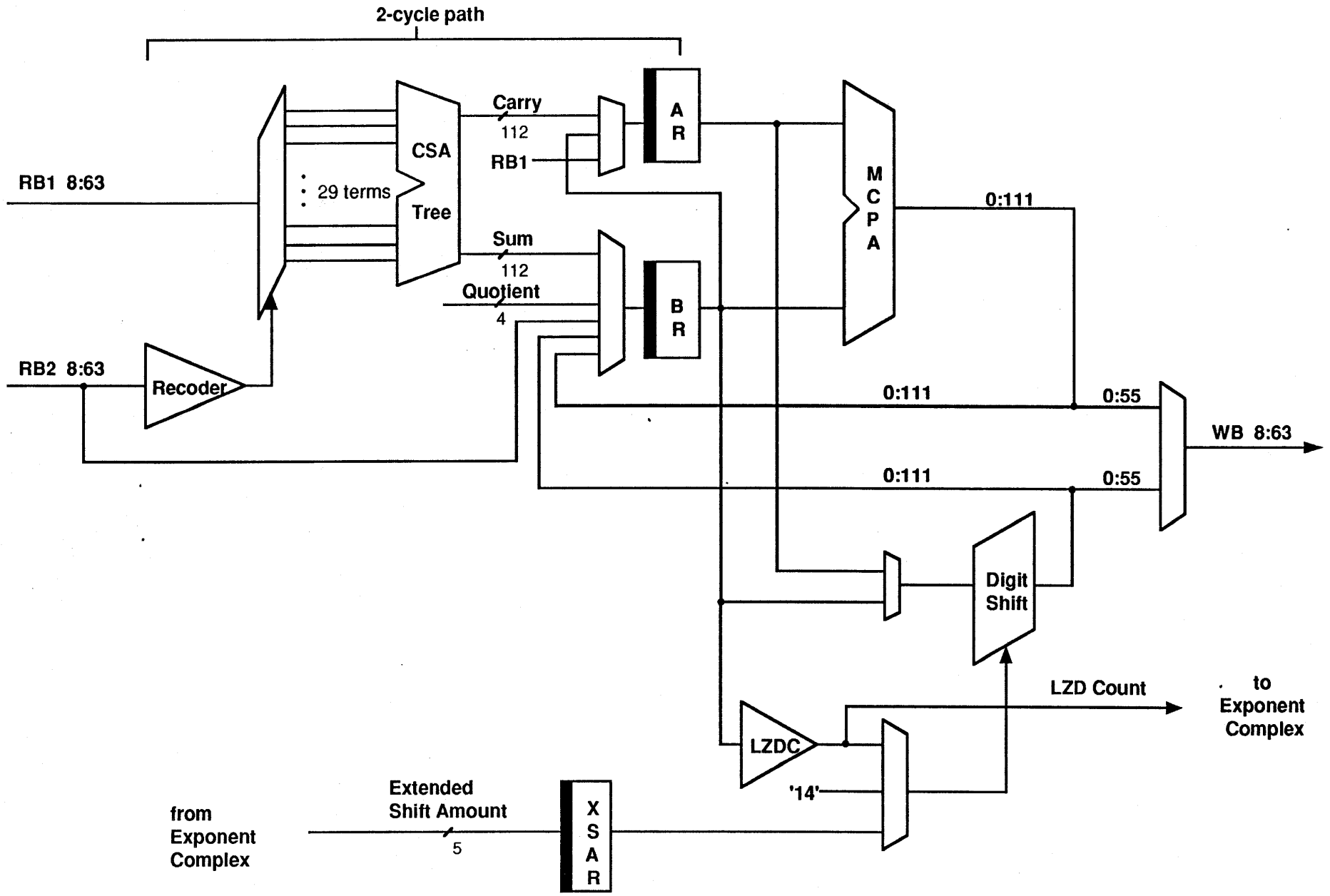
- **The adder has a latch point in the middle.**
  - Sum w/o byte carries is computed and latched.
  - Then the carries are added in.

- **If needed a recomplementation is done.**

- **Shifter normalizes results.**
  - Based on the carry out and Leading Zero Digit Count.
  - LZDC is also sent over to the exponent complex.

Floating Point Multiplier

Rev. 1, 5/91



Floating Point Multiplier

Multiplies

- **POO requires various combinations. Examples include ...**
  - Short x Short = Long
  - Long x Long = Long (truncated)
  - Long x Long = Extended
- **Discussion focuses on L x L = E. Other flavors are similar.**
- **Like Fixed Point multiply, but bigger.**
  - RB2 Recoded (modified Booth's alg) to select __ different ___ bit multiplicand terms.
  - CSA tree adds these to generate Carry and Sum terms, _____ bits each.
  - Multiplier CPA adds Carry and Sum. Propagate done by "brute force" in 3 levels:
    1. Generates lots of common terms (e.g. consecutive strings of propagates).
    2. Calculates, for each pair of digits, the carry-in from the lower to the higher one.
    3. ORs in, for each digit, all carries from lower digits.
- **If no Leading Zeros, can send out over WB right away**
  - Send the high half immediately.
  - Latch result into BR then shift by 14 to send the low half onto the WB.
- **If there are leading zero digits ...**
  - Latch result into BR.
  - Shift left based on LZD Count.
  - Send out result (same process as above).

Extended Adds

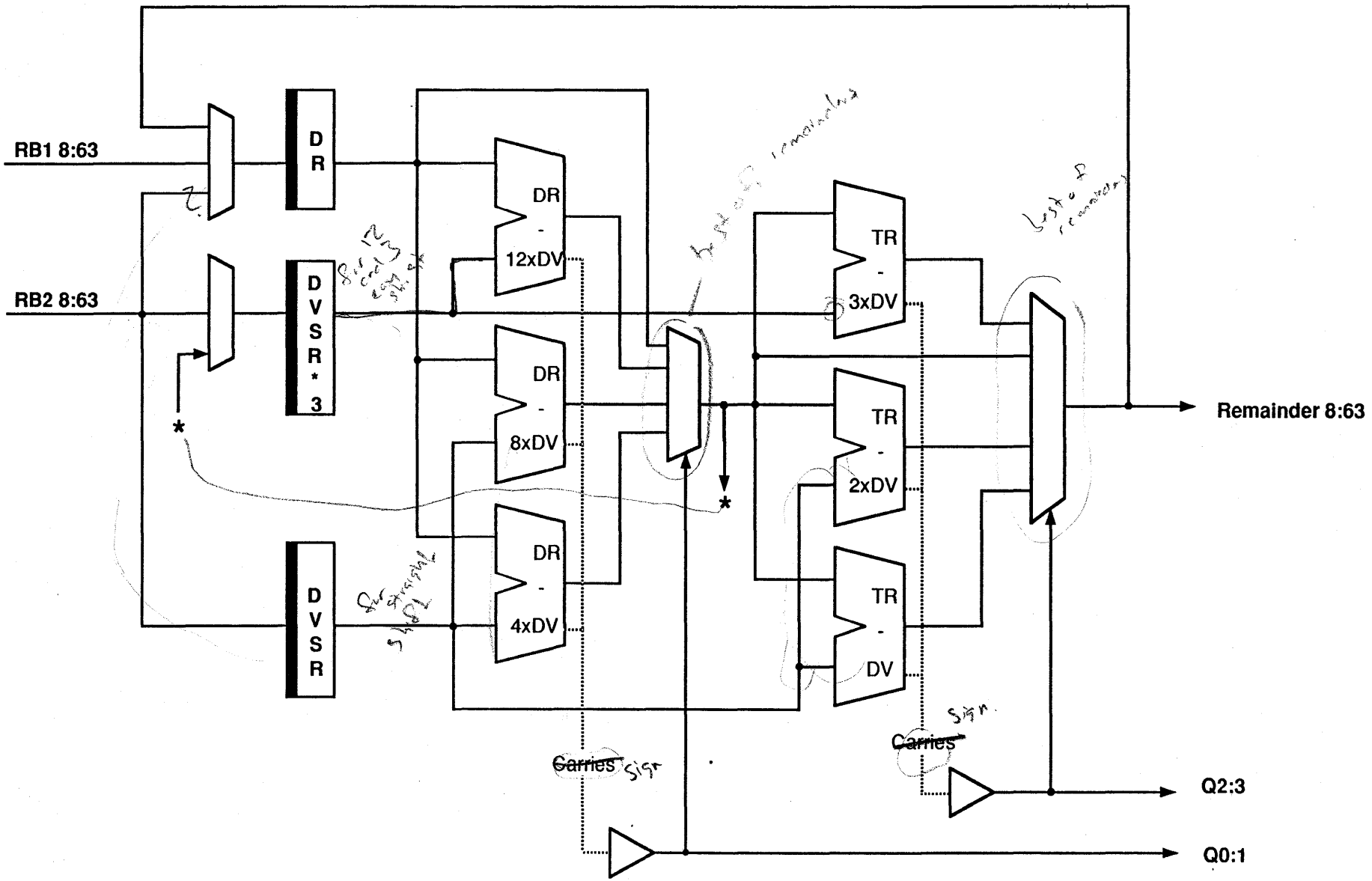
- Load operands into AR and BR.
- Using the Digit shifter, shift the fraction that has the smaller exponent by the Extended Shift Amount from the Exponent Complex. This fraction is restored into BR, and the other is put into AR.
- Add and post-normalize the same way multiply is done.

Divides

- The divider sends over 4 quotient bits per cycle, which are shifted into the BR.
- When quotient is complete, normalization then proceeds as above.

Floating Point Divider

Rev. 1, 5/91



Floating Point Divider

- **Two stage divide:**
  - Do trial subtractions of 4, 8, and 12 times the Divisor. This determines Q0:1.
  - Do trial subtractions of 1, 2, and 3 times the Divisor (subtracting from the Temporary Remainder calculated from the previous stage). This determines Q2:3.
- **Send Partial Quotient (Q0:3) to Multiplier for accumulation.**
- **Load Remainder back into DR.**
- **Using DVSR*3 allows all subtractions to be done via shift and subtract. To load DVSR*3:**
  - First, DR is loaded with 16*RB2 and DVSR*3 is loaded with RB2 (RB2 has the Divisor).
  - These are sent through the -12*DVSR subtractor. Since this subtractor assumes the DVSR has already been tripled, it's designed to do X-4Y, which gives (16-4)*DVSR = 12*DVSR.
  - This is shifted right 2 bits and loaded into DVSR*3 the next cycle.
  - Meanwhile, DR and DVSR are loaded with unaltered RB1 and RB2 values.

- This page intentionally left blank -

Decimal

Decimal Data Formats

Zoned Format

Packed Format

Z = Zone
D = Decimal Digit
N = Numeric
S = Sign

Each square represents 4 bits

Decimal Data Formats

2 formats, Zoned and Packed. Both are:

- Based on strings of bytes, each containing two 4-bit fields.
- Variable length → only used in SS ops.

Zoned

- First field is called Zone. This can be anything.
- Second field is called Numeric. Often it's a decimal digit.
- In the rightmost byte, the zone may be a sign digit.
- This format is set up for EBCDIC data manipulation. For example, decimal numbers in EBCDIC all have the same zone field value, and the numeric field contains the binary representation of the digit.

Packed

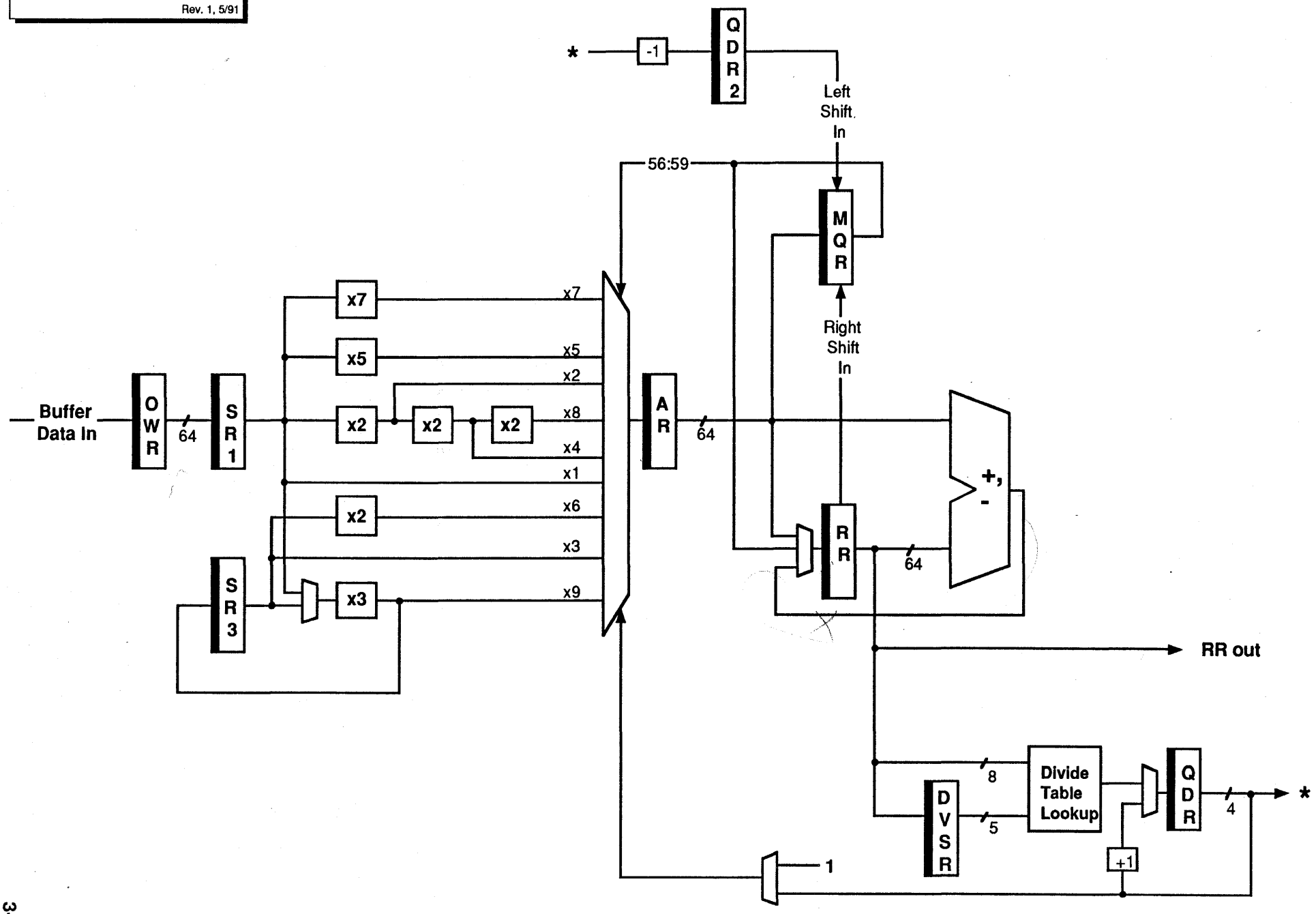
- String of decimal digits, terminated by a sign code.
- Digits must be 0-9. Sign is A-F, where low order bit is the actual sign bit.
- ➨ This is the format used by all arithmetic decimal ops.
  - Add, Subtract, Multiply, Divide

Some instructions are provided that convert between these two formats:

- Pack: converts from Zoned to Packed. Basically just strips out the zones.
- Unpack: converts from Packed to Zoned. Inserts F into Zone, making it EBCDIC.
- Edit, Edit & Mark: Very hairy. Converts from Packed to Zoned and allows lots of modifications on the way. Masochists are referred to Chapter 8 of the POO.

Decimal Unit

Rev. 1, 5/91



Decimal Unit

- **Basic elements include:**
  - The OWR (note that it's only fed from the buffer, since all OPs are SS).
  - Scratch Registers 1 and 3. SR3 is loaded with 3xSR1.
  - Digit Multipliers producing multiples of SR1 from 2 to 9.
  - Adder Register.
  - Result Register.
  - Adder/subtracter - operates on the RR and the Adder Register.
  - Multiplier/Quotient Register.
  - NOTE: data paths are all 8 bytes wide.

- **Addition**
  - Load the AR and RR with the operands. Then add into the RR.

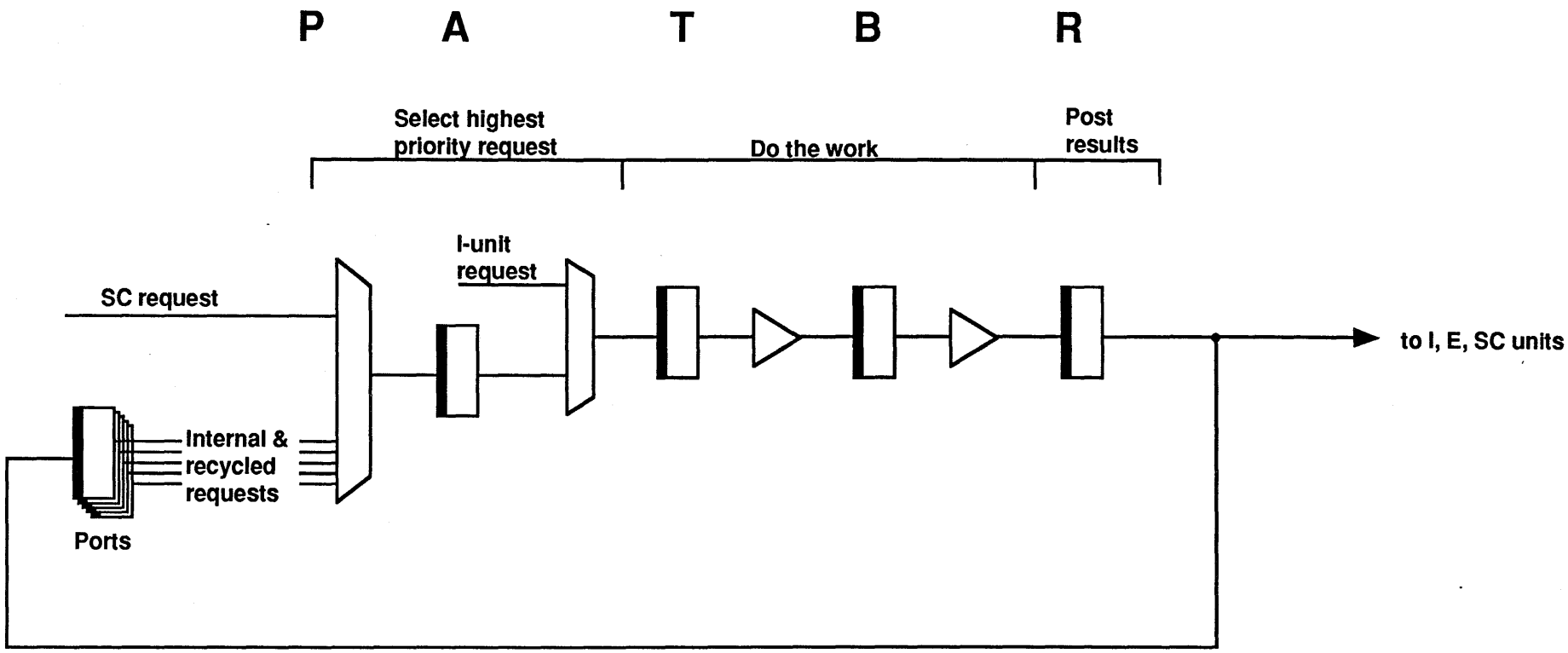
- **Multiplication**
  - Per POO: Multiply Decimal allows a Multiplier of ≤8 bytes and a Multiplicand of ≤ 16 bytes. Multiplicand must have enough leading zeros to ensure the result is ≤ 16 bytes.
  - Load the Multiplier into the MQR and the Multiplicand into SR1,3. Clear the RR, which will act as an accumulator.
  - Use MQR56:59 (60:63 is the sign) to select the proper multiple of the Multiplicand.
  - Add this to the contents of the RR and store the result back into the RR, shifting right as you do. The rightmost digit is shifted into the MQR and *its* rightmost digit is discarded.
  - Continue until done. The product is in the RR and MQR and can be read out over 2 cycles, if needed. A shifter (not shown) aligns the data before doing so.

- **Division**
  - From POO: DVSR ≤ 8 bytes, Dividend ≤ 16 bytes.
  - Load Divisor into SR1,3 and also put the upper 5 bits into DVSR. Load Dividend into RR.
  - Upper 8 bits of Dividend/Remainder and upper 5 bits of Divisor address a lookup table for a lower bound guess at the quotient digit. (Assumes remaining Divisor bits are _____ and Dividend bits are _____.)
  - Based on this guess, select a Divisor multiple and subtract from the Remainder.
  - Keep incrementing QDR and subtracting the Divisor times 1 until you get a carryout (ie. number goes negative).
  - At this point, don't load the RR, it has the correct Remainder. QDR2 has the correct Quotient, which is shifted into MQR.
  - Continue till done. MQR has the Quotient, RR has the Remainder.

S-unit

Basic S-unit Concept

Rev. 1, 5/91



Request:

Opcode	Address			Flags			
Opcode	Logical	(Physical)	(Misc)	Key	DIM ID	V/R	Misc

Basic S-unit Concept

- **Pipe has 3 basic stages:**
  - Select the highest priority request, including internal and external requests. Split across P and A cycles.
  - Do the work for that request. Split across T and B cycles.
  - Post the results. Done in B and R-cycles.

- **580 had just PBR. Apache and Sona added A and T for timing reasons.**

- **A request includes everything needed to complete processing, including:**

  Opcode (~150 of 256 used):

  I-unit: Fetches, stores, branch, SC ops (e.g. XSU stuff), TLB maintenance, register loading, misc.

  SC: Move-in flows, move-out flows, key ops, misc.

  Internal: TAG maintenance, TLB maintenance, translator flows, misc.

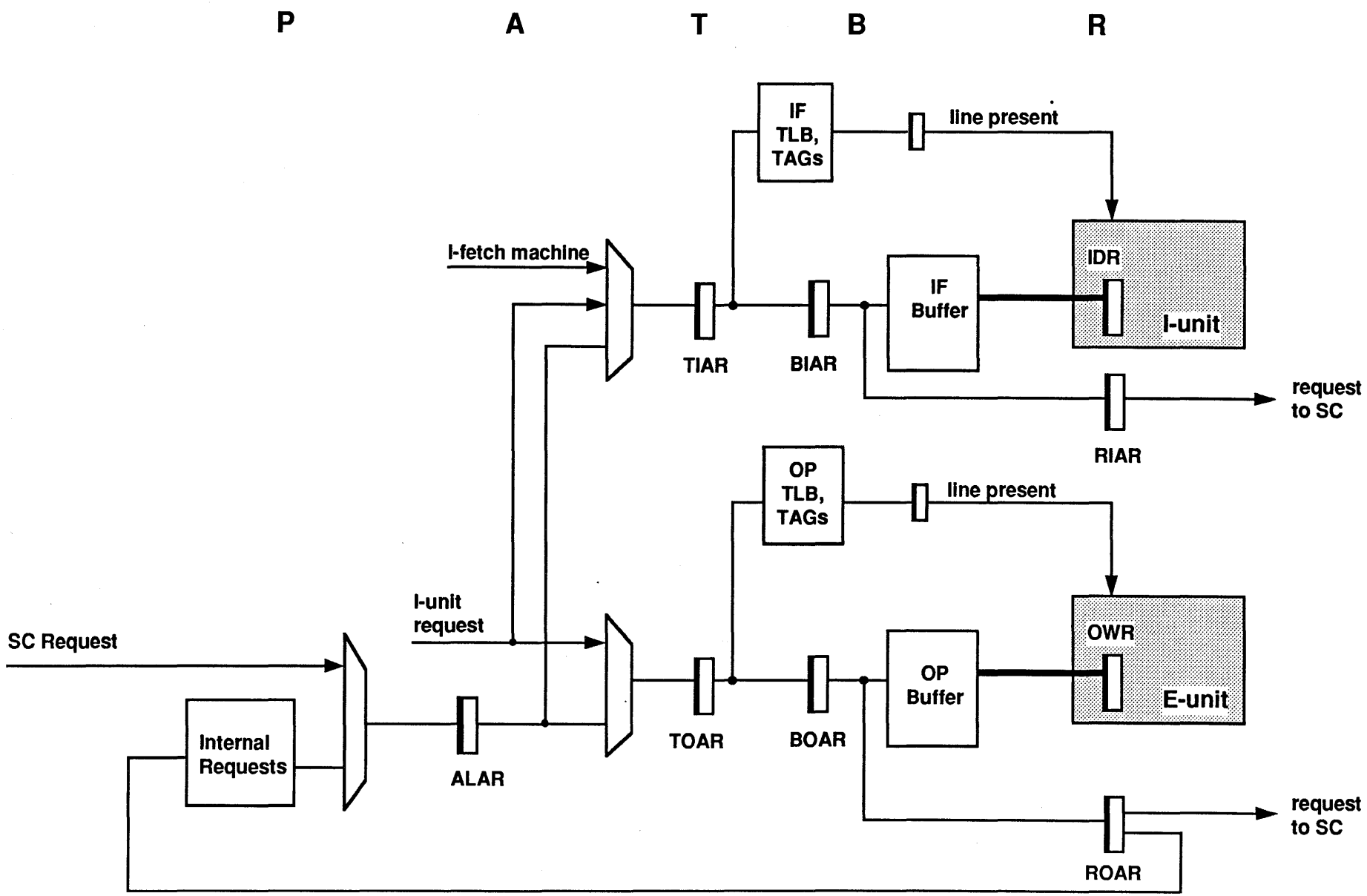
  Address:
  - Logical (called Effective in I-unit)
  - Physical (not supplied on I-unit requests)
  - Misc (STD)

  Flags:
  - Key
  - Address Dimension
  - Virt/Real
  - Others

- **Ports store the request for recycling.**

S-unit Pipes

Rev. 1, 5/91



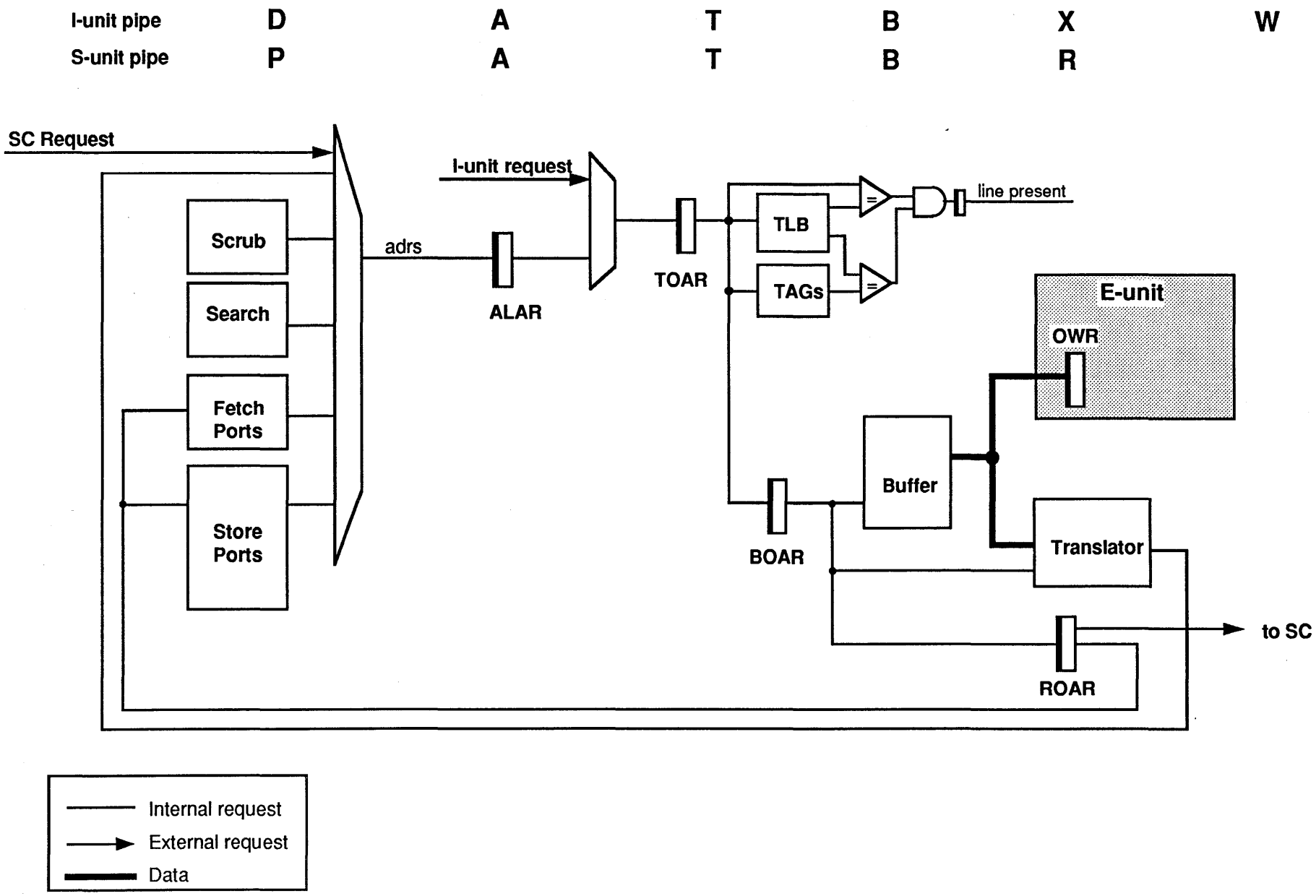
S-unit Pipelines

- **S-unit has two parallel pipelines, IF and OP.**
  - Each has its own TLB, TAGs, and Buffer.
  - OP sends data to the _____, IF sends data to the _____.
  - Pipelines are free-running; incomplete requests recycle until complete.

- **Two common OP pipe requests from the I-unit are Fetches and Stores.**
  - Much of the S-unit is tailored to these operations.
  - Fetches read data out of the buffer and into the OWR.
  - Stores have two parts.
    * The *store* flow reads the data from the buffer and into the OWR.
    * The *write* flow writes data (sent from the RR) into the buffer.
    * The store flow of a store is handled a lot like a normal fetch.

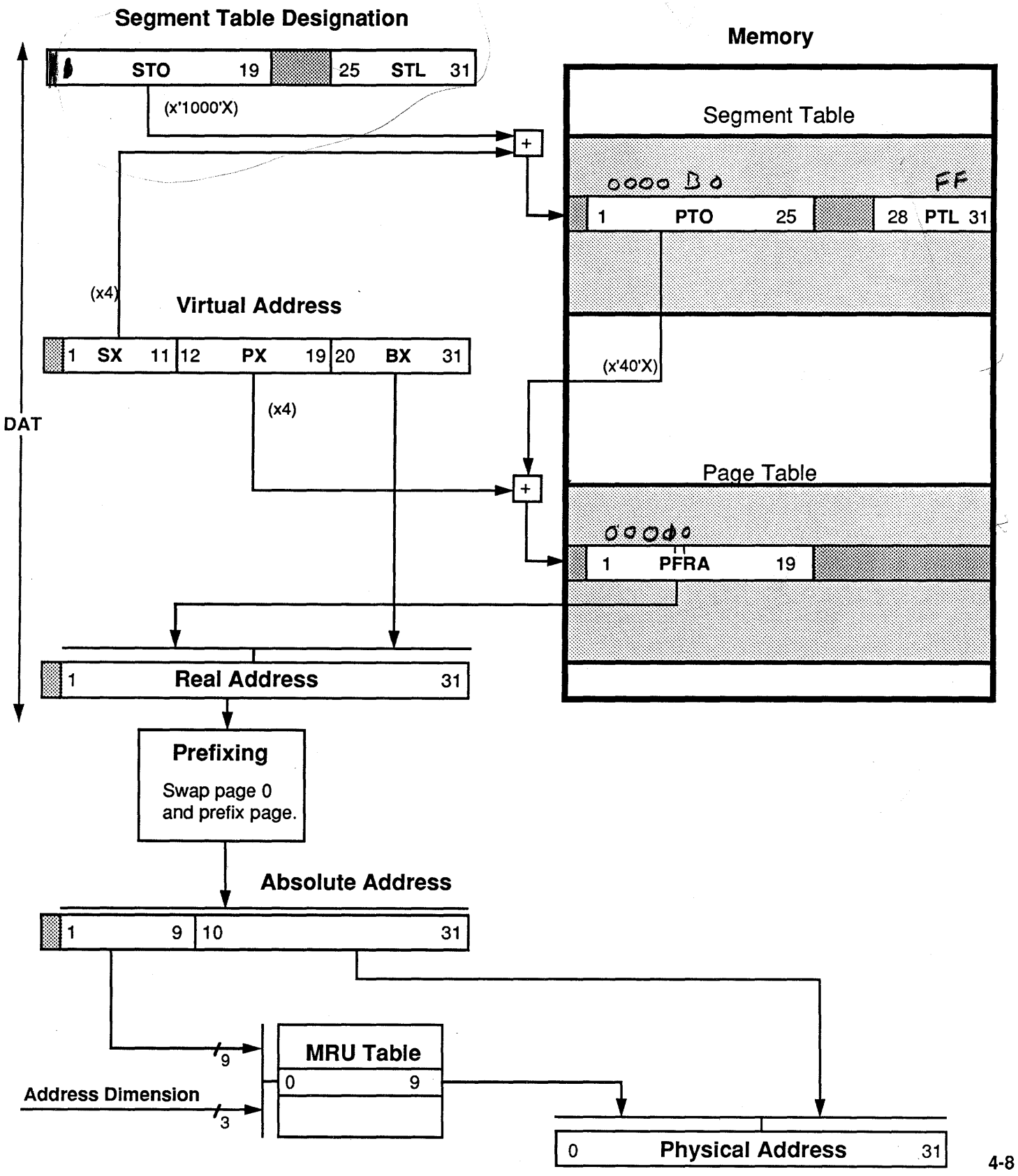
S-unit Basic Blocks

Rev. 1, 5/91



S-unit OP Pipe Basic Blocks

- **TAGs, TLB**
  - used to determine if line is present in the cache.
- **Buffer (a.k.a. cache)**
- **Translator**
  - does Virtual to Physical Address translations.
- **Fetch Ports**
  - contain fetch requests until they complete.
- **Store Ports**
  - contain write flows (of stores) until they complete.
- **Search Machine**
  - does background TLB maintenance.
- **Scrub Machine**
  - does background searches for single-bits errors.
- **SC Requests**
  - path used by SC to move data into and out of the buffer.
- **I-unit Request Processing sequence:**
  1. Requests priority in the A-cycle.
  2. If granted, TAG and TLB match done in the T. Buffer accessed in B.
  3. If line present, status valid posted and data clocked into the OWR.
     * Status Valid is a key signal from the S-unit. Indicates request completion, even for requests that don't return data. Lack of Status Valid leads to _____ in the I-unit.
  4. Otherwise, request is loaded into a fetch port.
  5. Later, the fetch port requests priority into the P-cycle.
  6. If granted, it contends for A-cycle priority and continues as from 1.



Address Translation

Dynamic Address Translation

- Maps Virtual Address to Real Address on 4K boundaries.
- IBM defined. Enabled by a _____ bit.
- Uses 2-level lookup of tables stored in memory.
  - Segment Table Origin 0:19 (left justified) points to beginning of segment table.
  - VA1:11 (for _____ segments) indexes into segment table in 4 byte increments.
  - Segment table entries are 1 word. Bits 1:25 form a Page Table Origin and (left justified) point to a page table.
  - VA12:19 indexes into the page table in 4 byte increments.
  - Page table entries are 1 word. Bits 1:19 form the Page Frame Real Address (i.e. bits 1:19 of the Real Address).
  - The PFRA is then used in place of the high order 19 bits of the address.
  - VA20:31 = RA20:31. No translation done on these bits.

Prefixing

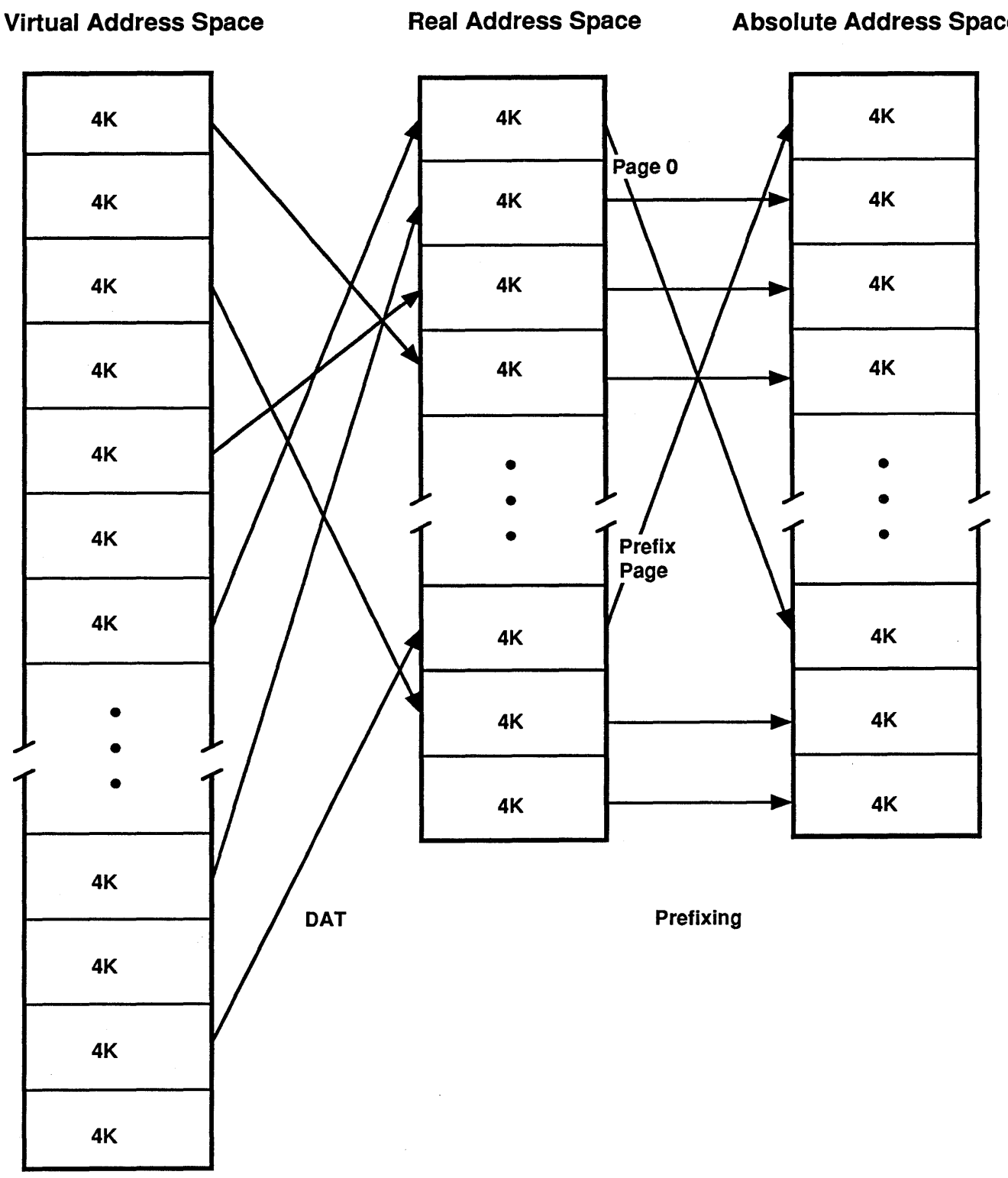
- Maps a Real Address to an Absolute Address.
- Also IBM defined.
- Swaps the prefix page (pointed to by a prefix register) with page 0. All other page addresses are unchanged.
- Allows each CPU's page 0 to point to a different address in memory.

Main Store Reconfigurable Unit Table Lookup

- Maps an Absolute Address to a Physical Address.
- Amdahl defined. Implemented in dedicated RAM.
- AA1:9, along with the Address Dimension, index into a table which provides PA0:9. AA10:31 are unaltered.
- Allows Domains (each Domain is in a different Dimension) to map to its own chunk of memory, and to give an Addressing Exception if a Domain tries to go outside its bounds.
- Also used to reconfigure Main Store in 4M chunks.
- Called MRU Table or MRUT.

Page Mapping

Rev. 1, 5/91

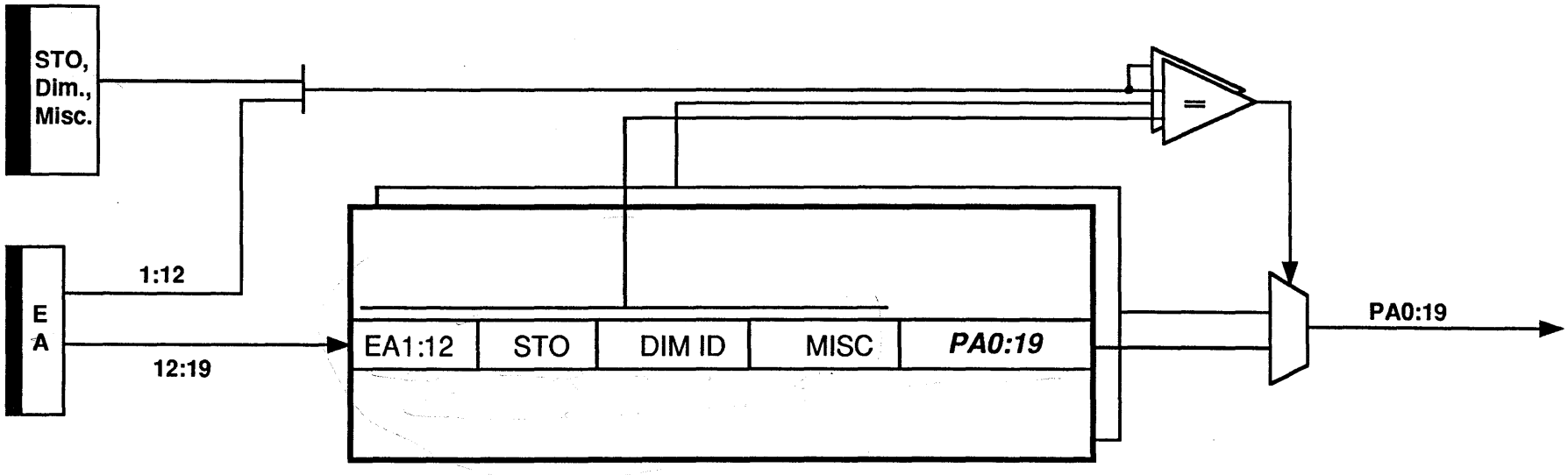


STD	VA
00003001	00903A5F

MAIN MEMORY

3000	34420A4A
3004	4820BC8A
3008	BB836B54
300C	34420A4A
3010	9BA6473B
3014	C8574387
3018	578AECEF
301C	34420A4A
3020	00003D41
3024	00003054
3028	FF738A63
302C	0000349A
3030	A7D8F9EE
3034	00003442
3038	34420A4A
303C	56734E2A
3040	4820BC8A
3044	FFF00123
3048	47584FDA
304C	00489AE7
3050	15283754
3054	ADFFD456
3058	47386954
305C	98473859
3060	A6734251
3064	39CD70F3
3068	A8DF3490
306C	32859604

* note: all numbers are in hex

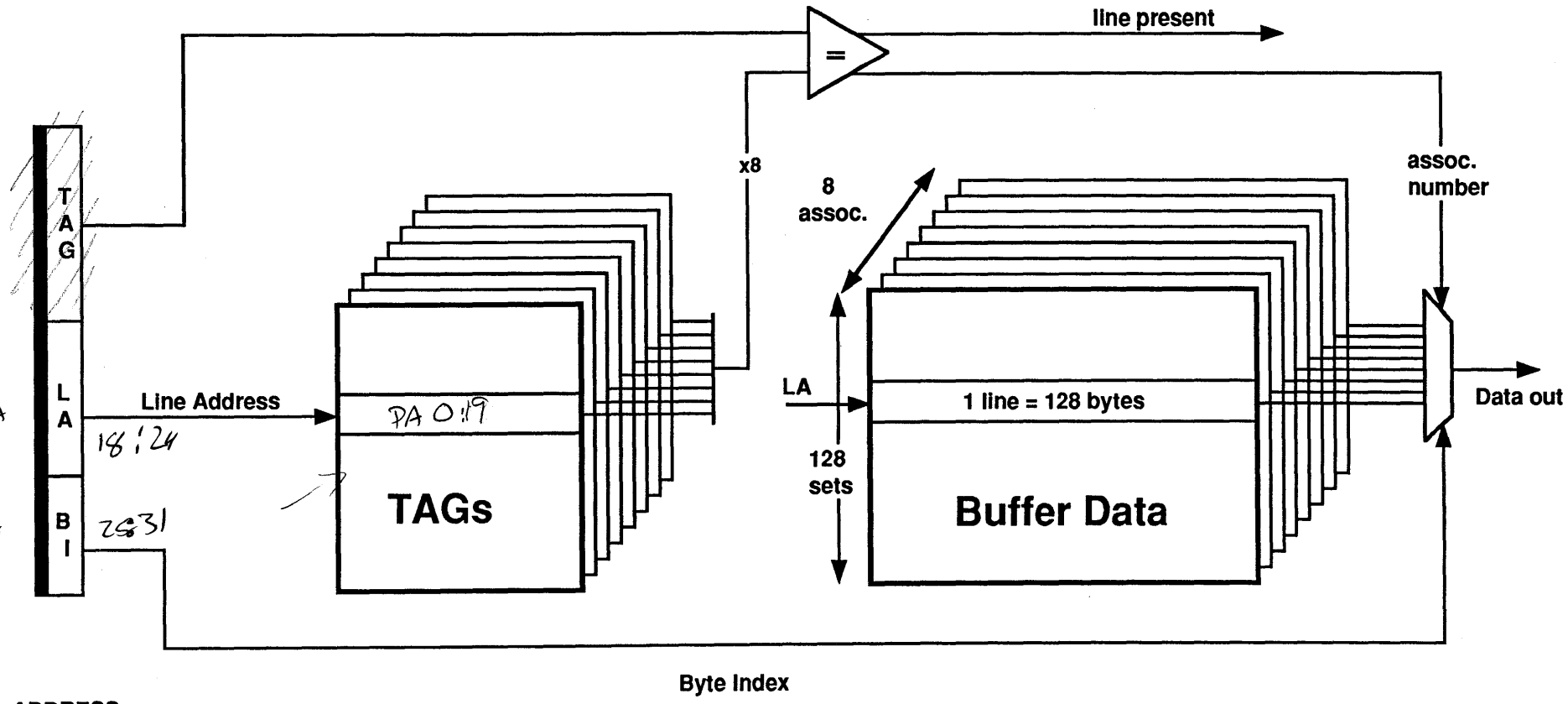


Translation Lookaside Buffer

- Holds recently used translations.
- 256 sets x 2 associativities.
- Addressed by _____.
- Match against EA1:12, STO, Address Dimension, various flags.
- TLB "data" is _____.
- Real to Physical address translations are also stored in the TLB.

TAG, Buffer Organization

Rev. 1, 5/91

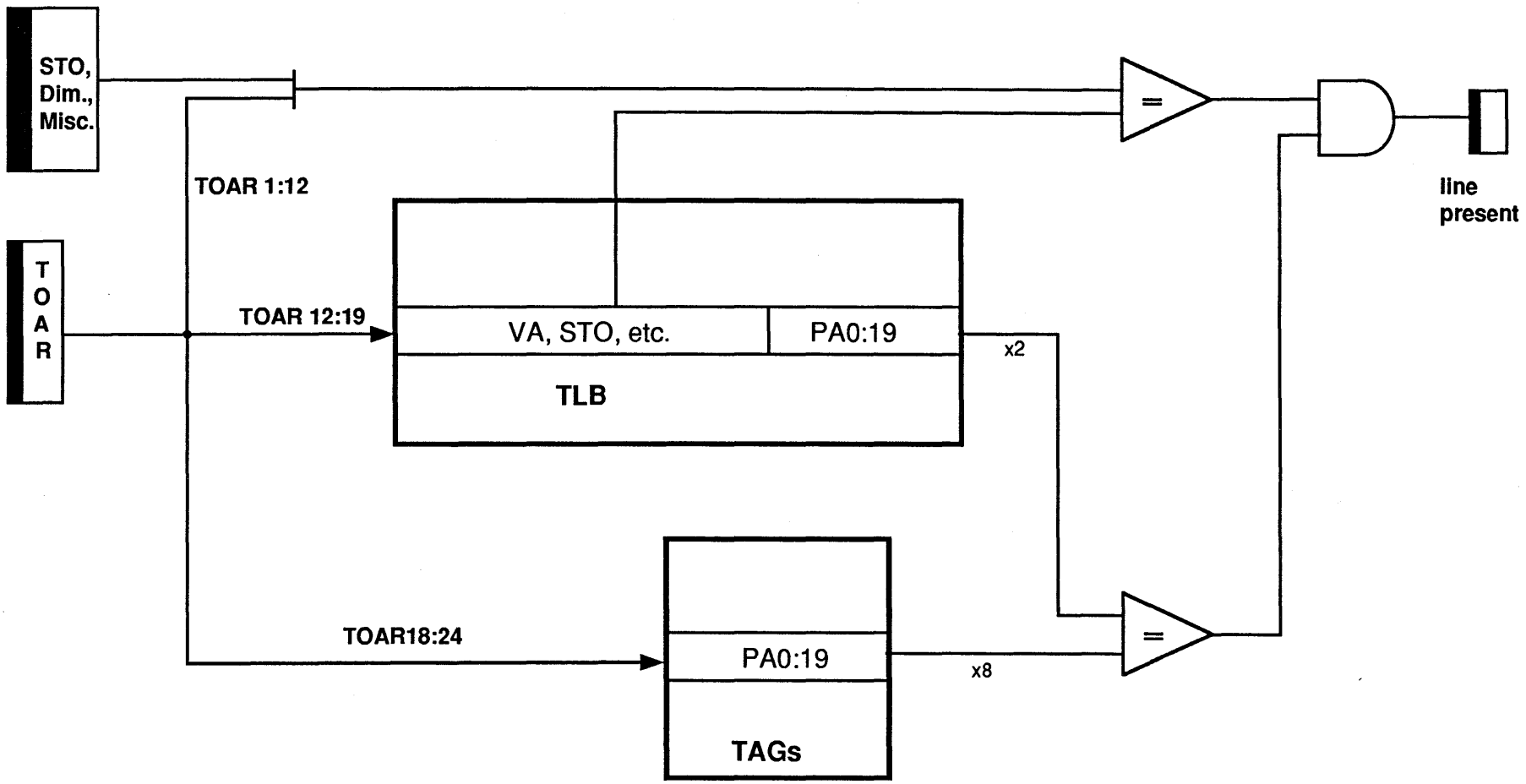


Buffer Organization

- **128 sets x 8 associativities**
- **128 bytes lines.**
- **TAGs contain _____.**
- **Addressed by _____.**

OP TAGs/TLB

Rev. 1, 5/91



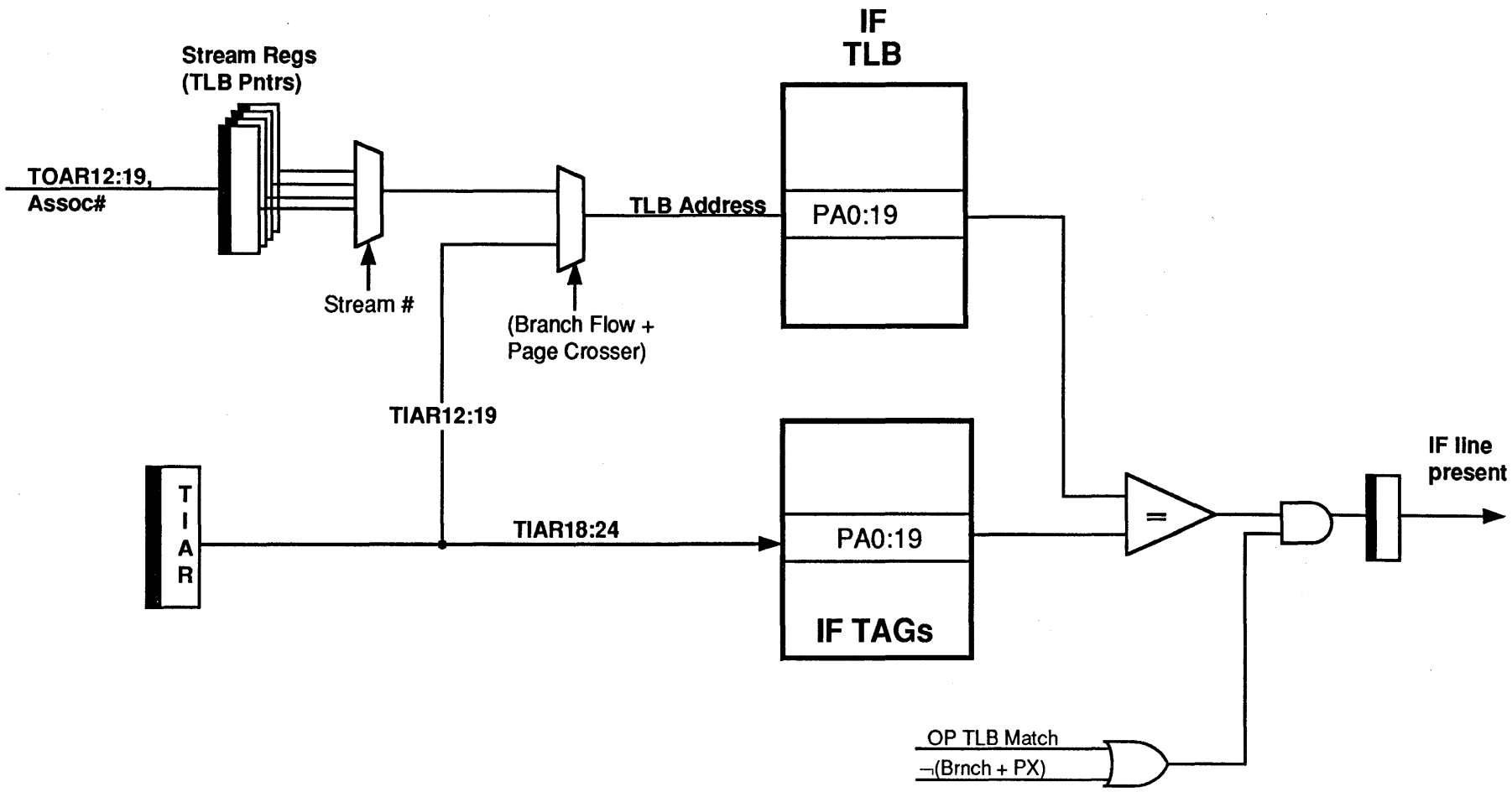
OP TAGs/ TLB

- **TAGs/TLB used to determine line state. Possible states are:**

Translation in TLB?	Line in Cache?	TLB Match?	TAG Match?
Yes	Yes	Yes	Yes
Yes	No	Yes	No
No	Yes	No	NA
No	No	No	NA

IF TAGs/TLB

Rev. 1, 5/91



IF TLB

- **IF TLB is a copy of the PA portion of the OP TLB.**
  - When creating TLB entries, Translator writes to both OP and IF TLB.

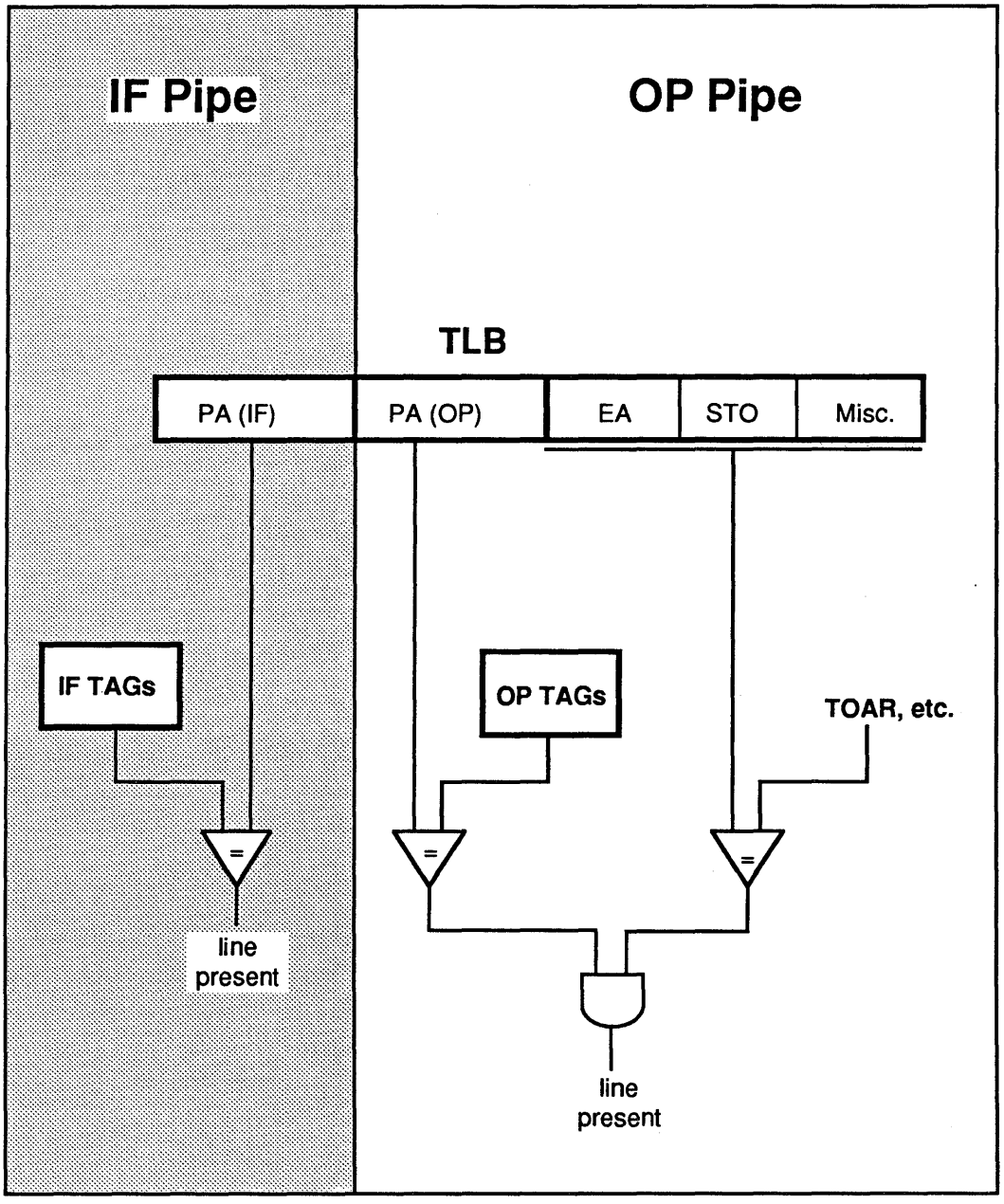
- **Stream register holds TLB address for current page.**
  - Current page = page containing current instruction address.
  - Loaded/validated by accessing OP TLB (via OP pipe) on branches and when instruction stream crosses 4K page boundaries (a.k.a. *IF TLB Validate* flow).
  - This validation flow accesses the OP TLB to do a match to make sure a valid translation is in the TLB.
  - If the validation flow gets a match, the matching location (TLB address and associativity) is saved in a stream register.
  - All subsequent IF flows for this stream use this "pointer" in the stream register to just read the PA out of the TLB and use it for TAG match; TLB match is implicit.
  - Note that this means reading out the same entry from the IF TLB over and over until you branch or cross into a new page.

- _____ **stream registers allow for late branch decisions.**

```
                Stream  Register  Timing
IF TLB Vldte (OP,IF)    A T B R
OP TLB Match              |-|
Stream reg loaded         |----->
Seq IF - IF Pipe        A T B R
Access TLB                |-|
TAG Match                   |-|
```

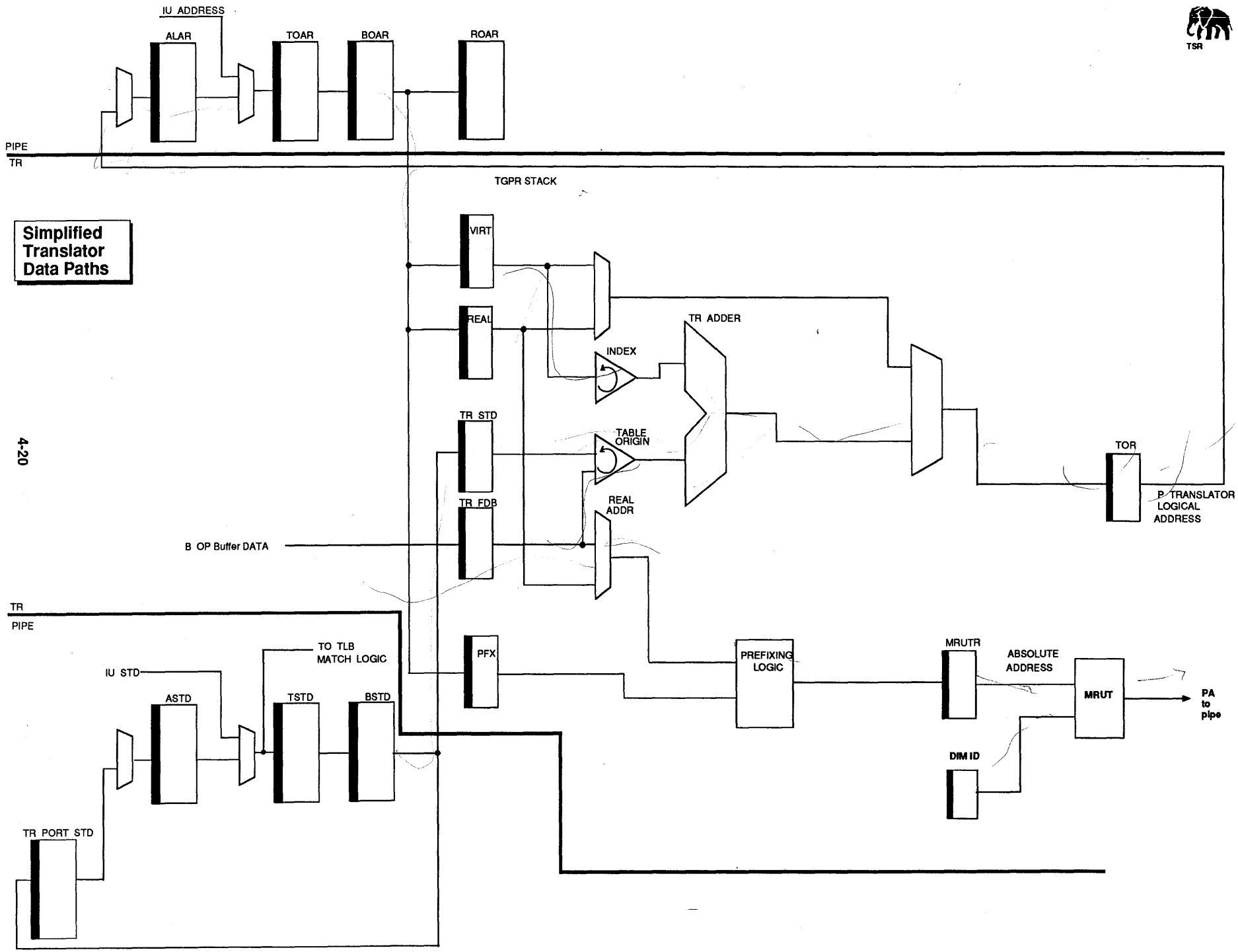
IF TLB - Another Perspective

Rev. 1, 3/92



IF TLB - Another Perspective

- **Key Points:**
  - TLB entry includes 2 copies of the PA field.
  - One copy of the PA is accessible from IF, the other from OP.
  - Each pipe uses the PA for its TAG match.
  - To make a new TLB entry the translator has to go down IF (as well as OP) to write the IF copy of the PA.
  - The OP pipe does an explicit TLB match each cycle, whereas the IF just does it on an IF TLB Validate flow, then remember the results.



Translator Data Paths

The basic translation algorithm is:

1. Load VIRT on a TLB miss with the address that's in the pipe.

2. At the same time, load TR STD with the STO associated with the TLB miss.

3. Add the Segment Index from VIRT to the STO to generate a segment table entry address in the TOR.

4. Send the TOR down the pipe (when granted priority) to fetch the segment table entry.

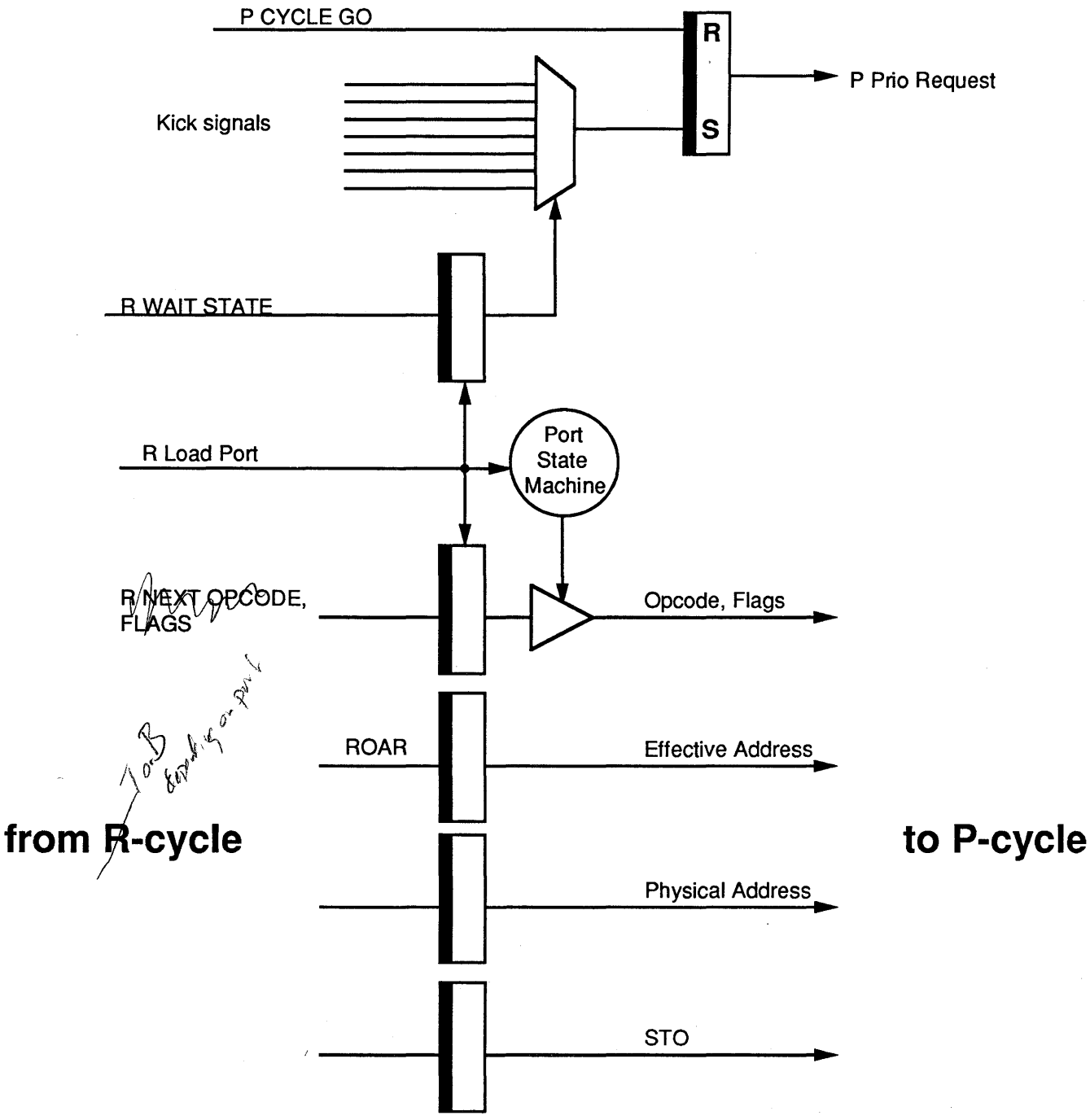
5. Load the STE (i.e. the Page Table Origin) into TR FDB.

6. Add the Page Index from VIRT to the PTO, and send this address down the pipe to fetch the PageTable Entry.

7. Load the PTE (i.e. the Page Frame Real Address) into the TR FDB. Send it through prefixing and MRUT to get a physical address.

8. Take one last flow down the pipe and use the VA, STO, and PA to make a new TLB entry.

- Note: if table entry fetches (which use Real Addresses) get a TLB miss themselves, load their address into REAL, do Prefixing and MRUT, and make a TLB entry. Then continue with the original translation.



General Port Structure

- This is a general picture of a port. Actual ports may be a subset of this.

- When a Port is loaded, Addresses (EA, STD, PA) are clocked into registers.

- Opcode and flags are also clocked in, but they may differ from the original versions, depending on the results of the flow just completed.

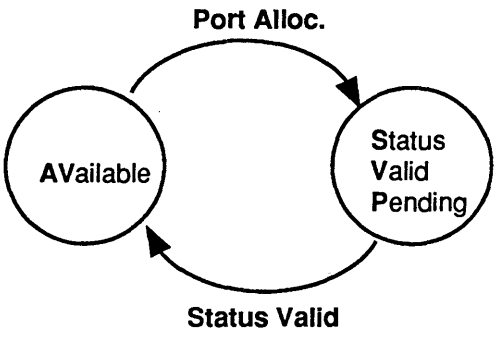
- A state machine keeps track of the port state, including some external events, and may even modify the opcode if required by these external events.

- Wait state is loaded to indicate what the algorithm is waiting on (e.g. on a TLB miss, a Fetch request would go into a Translator Wait state while the translation is being done). This controls a selector that monitors the various possible "kick" signals.

- Once kicked out of the wait, the port requests priority to the pipe.

Fetch Port State Machine

Rev. 1, 5/91



Fetch Ports

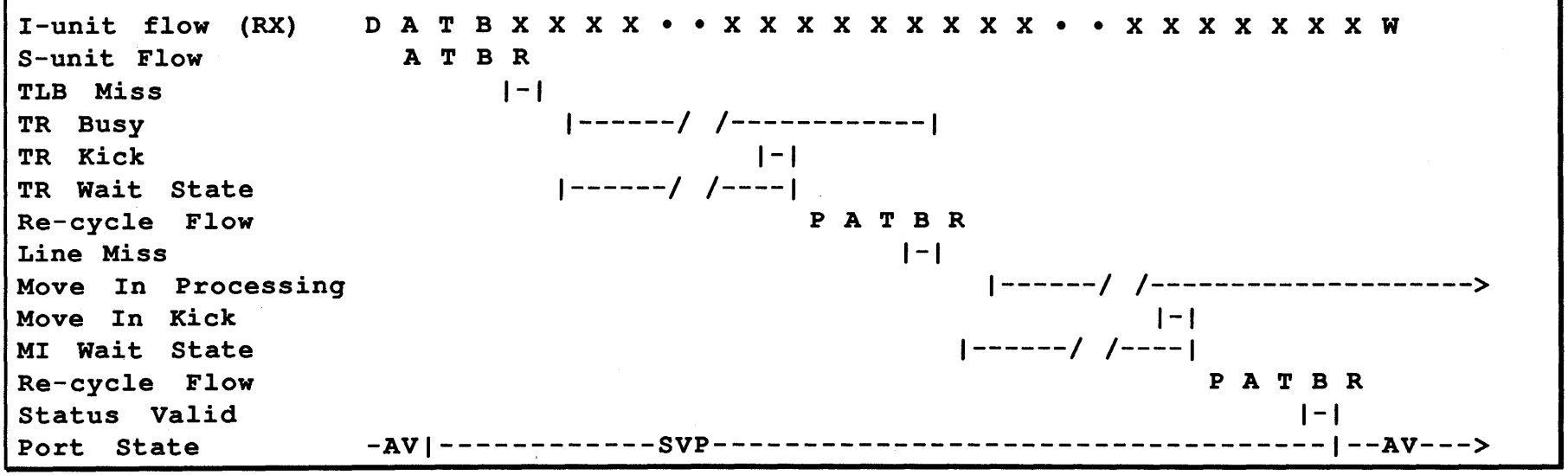
- Just two states, busy and available.

- Port goes busy (is "allocated") when I-unit request gets priority into A. If the external flow completes, the port won't actually be needed.

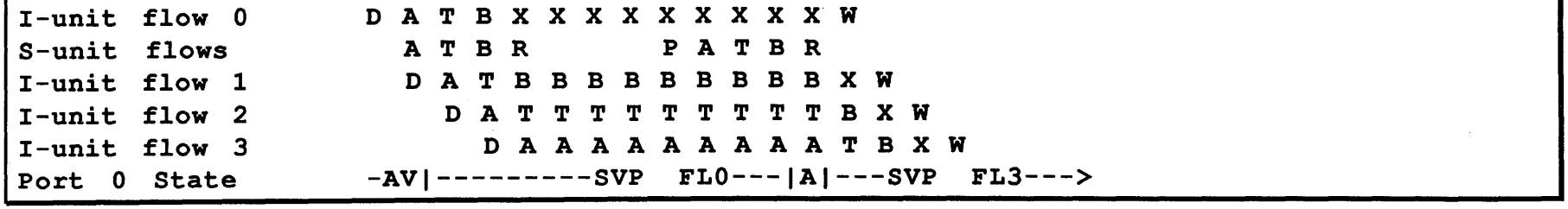
- An independent mechanism keeps track of the order of port allocation. The oldest request is called Top Of Queue. The TOQ request is the only one that's allowed to post Status Valid (i.e. send results) to the I-unit.

- Need _____ fetch ports for no-wait service.

Fetch (TLB and TAG miss) - Simplified



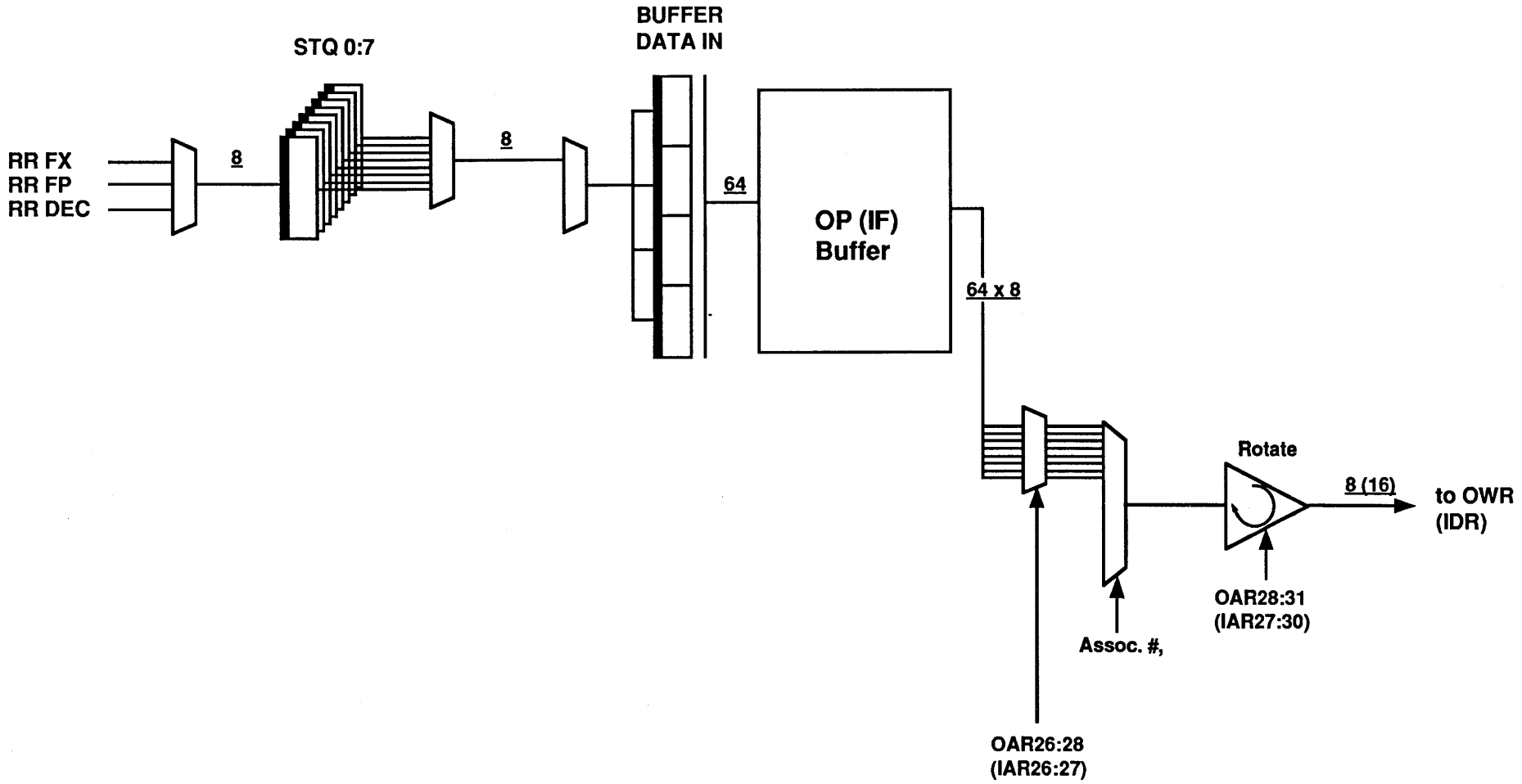
Multiple fetches - 1st one has line missing



- This page intentionally left blank -

Buffer Data Paths - partial

Rev. 1, 5/91



Buffer Data Paths (incomplete)

Output paths

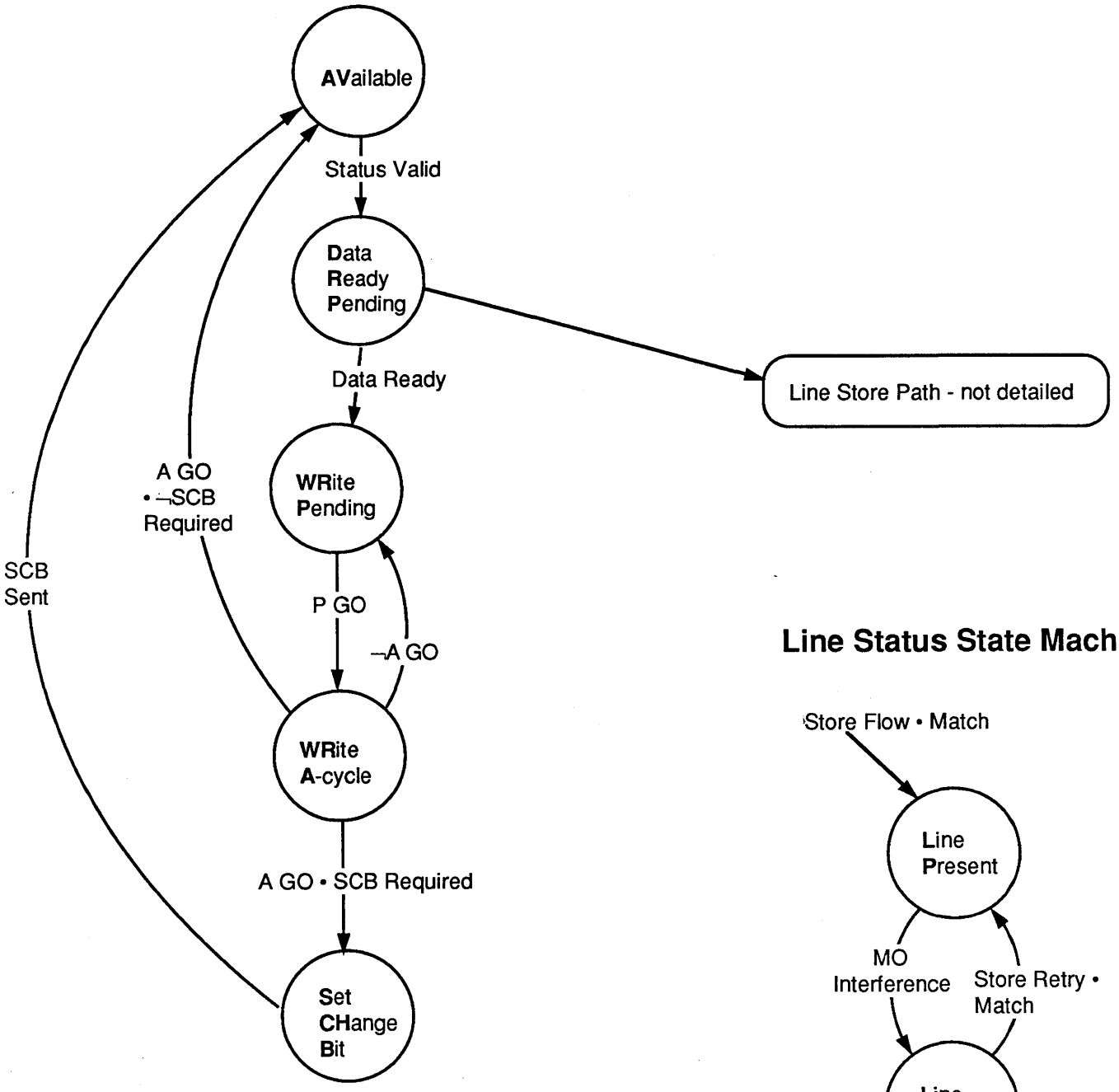
- 8 associativities of data.
- 64 bytes read out (bit 25 used in addressing buffer, though not TAGs).
- Low order address bits used to select the correct data, then align it to send into the OWR.
- Note IF differences due to 16 byte output path and halfword alignment.

Input Paths

- RR data clocked into Store Queue on Data Ready.
- Write flow eventually writes data from Store Queue into buffer.

Store Port State Machine

Rev. 1, 5/91

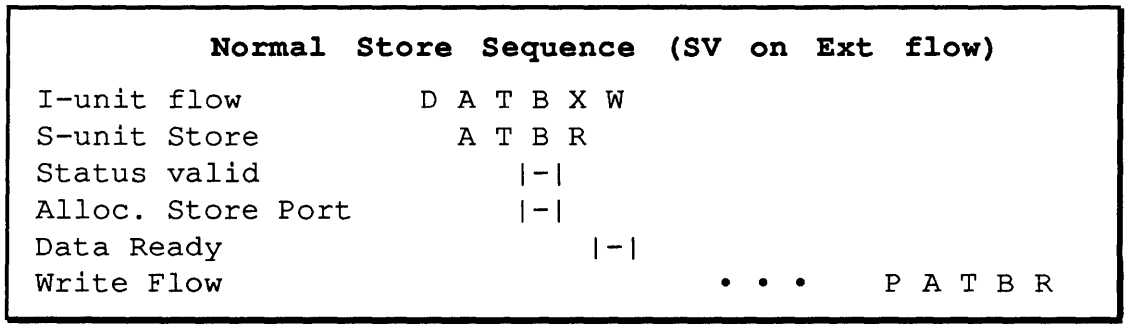


Line Status State Machine

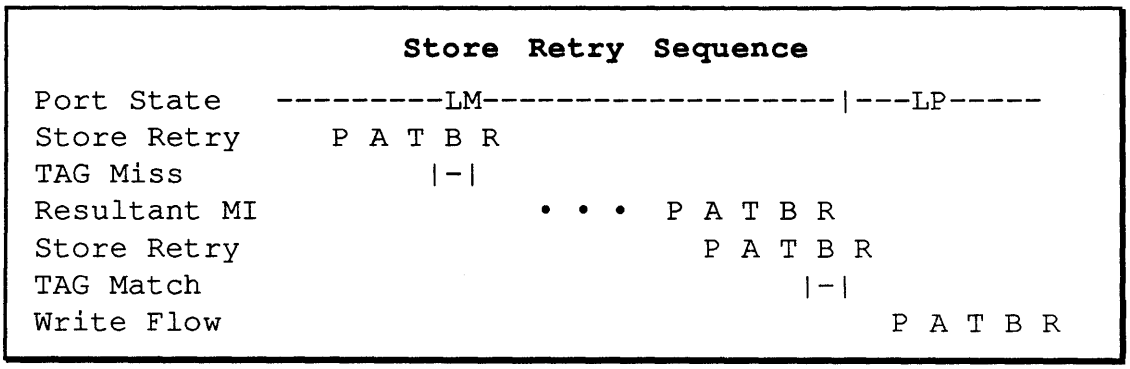
Store Port State Machine

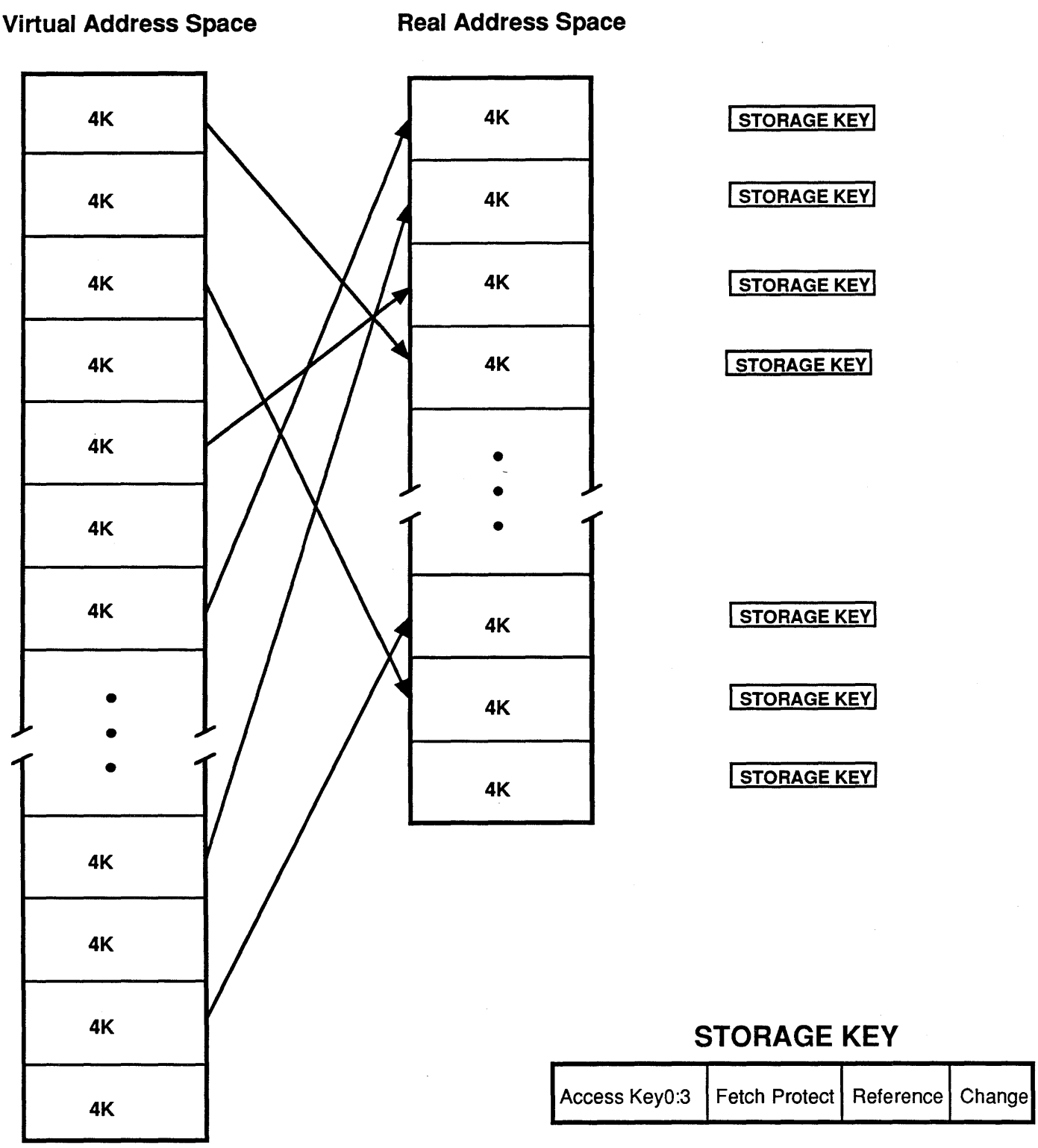
- **Tracks status of write portion of stores**
  - Allocated on SV of store flow. Responsibility passed from the fetch port to the store port.

- **Basic Store Algorithm**
  - Waits for Data Ready, then starts requesting priority.
  - Once write flow gets priority (both P and A), it's done.



- **Line Status State Machine**
  - Tracks presence of line in cache.
  - During MO's, address of line moved out is compared with addresses of pending stores. MO interference called on a match. Machine goes to Line Missing state.
  - On Line Missing, Store Retry flow initiated (a separate mechanism tracks priority grants).
  - Store Retry matches the PA from the Store Port with the PA in the TAGs to see if the line is in the cache. If not, a MI requested.
  - When the SR flow finally gets a match, the LS machine goes back to Line Present State.
  - Store-ahead: if the fetch data isn't needed (e.g. on a Store), you only need *TLB* match to post SV. If TAG Miss, allocate Store Port in Line Missing State.





STORAGE KEY

Access Key0:3	Fetch Protect	Reference	Change

Set Change Bit

- **Each 4K page has a Storage Key. Includes ...**
  - * 4 bit Access Key (often just called the Key):
    - Matched against a 4 bit key in the PSW.
    - Mismatches on stores cause a protection exception.
  - * Fetch Protect bit:
    - If this is a one, protection checking applies to fetches also.
  - * Reference bit:
    - Set when a storage reference is made to the page.
    - Used _____.
  - * Change bit:
    - Set when the page is modified.
    - Used _____.

- **System storage includes a key array associated with the MSU.**

- **A copy of the key is kept in the TLB for protection checking.**
  - Also tracks modified state of page (i.e. Change Bit).

TLB Contents

EA1:12	STO	DIM ID	P/P	Misc

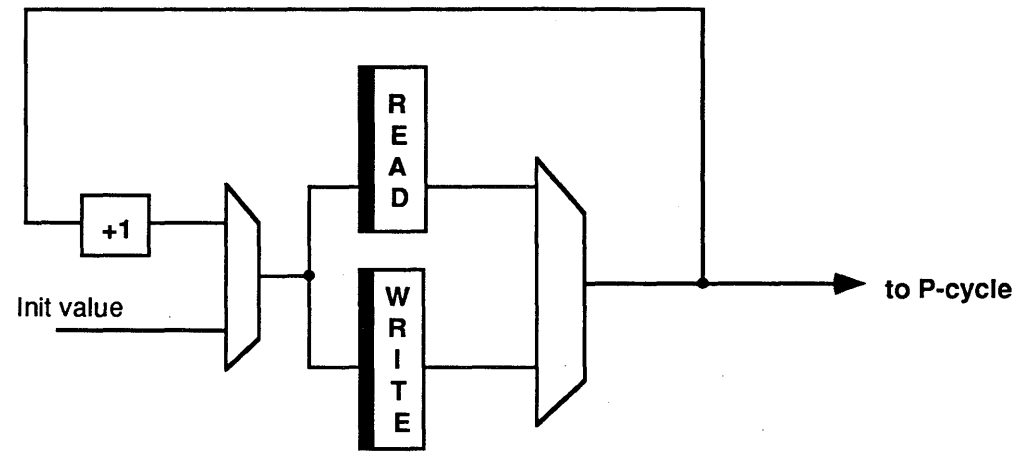
PA0:19	KEY0:3, C, FP

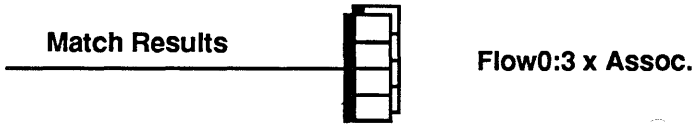
- **On Stores to a page with C=0, need to update TLB and send Set Change Bit message to SC.**

- **SCB state added to state machine to track pending SCB. Priority handled separately.**

Simplified Search Machine

Rev. 1, 5/91





Read Flow0	P	A	T	B	R				
Read Flow1		P	A	T	B	R			
Read Flow2			P	A	T	B	R		
Read Flow3				P	A	T	B	R	
Flow0 Match				⊢⊣					
Write Flow0					P	A	T	B	R

Search Machine

Architectural requirements

Purge TLB - invalidate all virtual entries in the TLB for the current domain.

Invalidate Page Table Entry - given PTO and PX, set Page Table and TLB entries invalid.

Set Storage Key - store new key value into Key Array at given Real Address..

PTLB Algorithm

- A Pre/Post latch is written into TLB entries when they're created.
- This same P/P bit is included in TLB match (i.e. match the P/P bit in the TLB with the P/P latch). Normally all entries will match the latch.
- PTLB toggles the latch and posts SV. All entries will now mismatch.
- The TLB is searched In the background for entries to invalidate, based on matching:

  _____

  _____

- New entries, and those that have been examined, have their P/P bit set to the new value so they'll match.

Background search implementation:

- Read flow reads TLB contents and matches them against appropriate search parameters (both associativities).
- Write flow writes entries to appropriate new state. Two write flows per read flow.
- Match results for 4 flows accumulated to help deal with pipe latency.
- The current R and W addresses are kept in separate registers.
- Note: Abandon TLB does same thing, but forces match on all searches.

IPTE Algorithm

- Search parameter is the PA (called Search Physical Address Match).
- SPAM inhibits status valid on fetch flows (a match indicates the fetch wants the TLB entry that has a pending IPTE).
- ___1___ read flows for 1MB segments.

SSK Algorithm

- Same as IPTE: search for PA; SPAM match on fetch flows causes the SSK key to be used for protection check in place of the TLB key.
- Have to search ___256___.

Scrub Machine (not detailed)

- Does background fetches looking for buffer single bits to clean up.
- Looks a lot like the Search Machine.

TAG Contents

Rev. 1, 5/91

TSR

Cycle Accessed

T	T	T	T	R

TAG Entry

PA0:19	Valid	Private	Modified	IF Pair, IF Pair Assoc. 0:2

1 per <u>associativity</u>

LRU Data
(R-cycle)

0>1	0>2	0>3	0>4	0>5	0>6	0>7	1>2	1>3	...	4>5	4>6	4>7	5>6	5>7	6>7

1 per <u>set.</u>

TAG Contents

- **Valid bit - TAG entry is valid.**

- **PA0:19**
  - page physical address.
  - matched against PA0:19 in TLB.

- **Private bit**
  - If '1', this is the only cached copy of the line (line is Private).
  - This CPU is allowed to modify the line.
  - If '0', this line is read only (line is Public).
  - System Controller is responsible for setting this bit correctly when moving the line in.
  - OP Cache is usually about 90% Private.
  - IF Cache is almost entirely Public (see IF Pair below for exception).

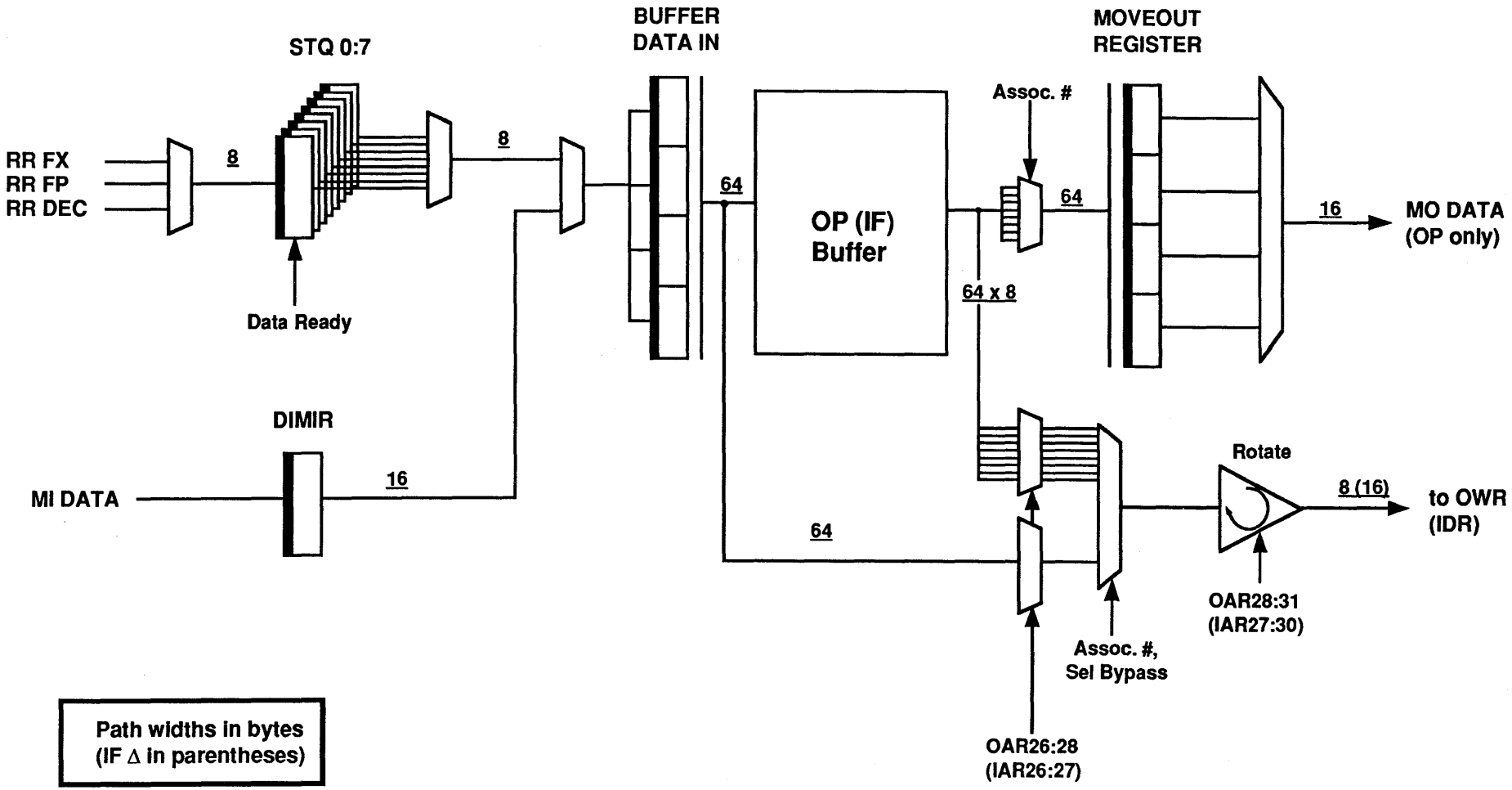
- **Modified bit**
  - If '1', the line has been modified since being moved into the cache.
  - If Modified, the line state must also be _____.
  - If Modified, need to back store to MSU eventually.
  - About 50% of Private lines get Modified.

- **IF Pair**
  - Means line is private in OP and IF has a copy at the same line address.
  - The Write flow of a Store will write both OP and IF copies.
  - Used when a line contains both operands and instructions. Prevents thrashing.
  - The Line Pair state is created by the SC when the line is moved in.
  - IF Pair Assoc. points to the associativity of the other half of the IF pair.

- **LRU data**
  - 1 bit for each pair of associativities in the set (covering all combinations).
  - indicates which associativity of the pair has been accessed more recently.
  - used to determine which assoc. is Least Recently Used (for replacement on Move-Ins).
  - one entry per *set.*

Buffer Data Paths

Rev. 1, 5/91



Buffer Data Paths

16 byte (1 QW) MI path from System Storage

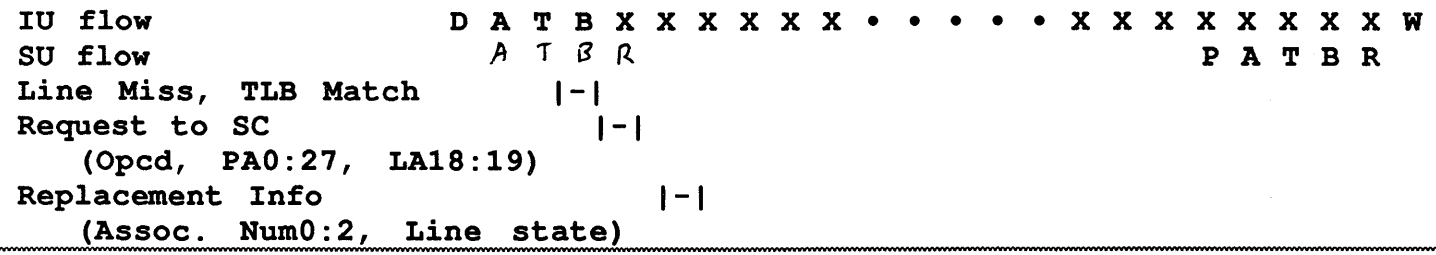
- 64 bytes (4 QWs) accumulated for buffer data-in.
- Bypass path from Data In Register.
  - Requested doubleword will be in first QW returned.
  - Can be bypassed to OWR while subsequent QWs are being accumulated.

16 byte MO path to System Storage

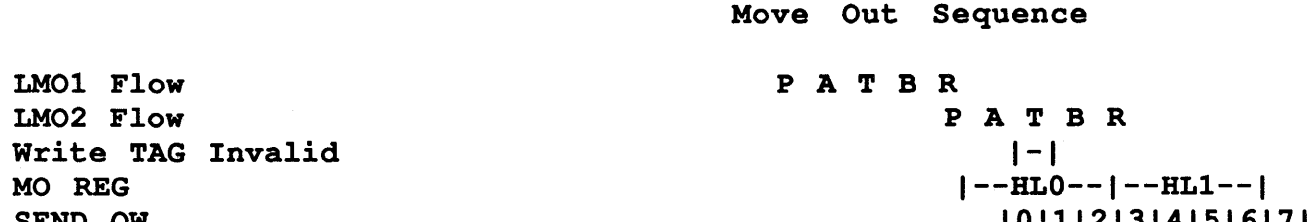
- 64 byte MO register latches data from selected associativity.
- Muxed out to System Storage over 4 cycles.

TSR

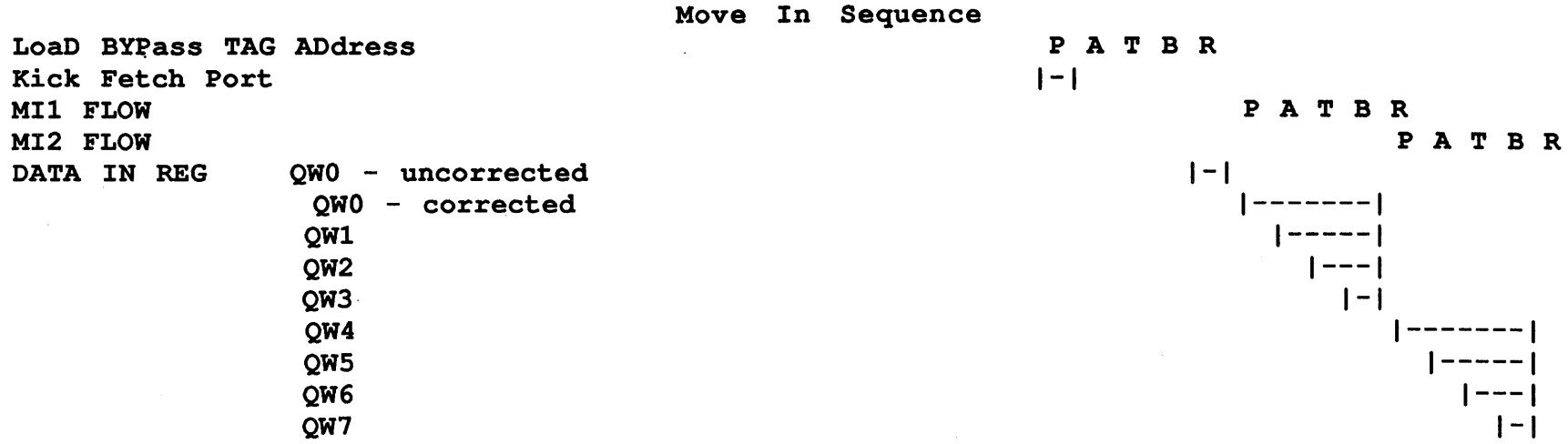
Line Miss Flow



Move Out Sequence



Move In Sequence



S-unit - Fetch w/Line miss

1. External flow

- Gets TLB match, giving us the PA.
- Gets TAG miss - the line needs to be moved in.

2. Send Move In request to SC

- In R-cycle, send:
  - opcode (e.g. Fetch Private).
  - PA0:27 (low order bits indicate which QW to move in first).
  - EA18:19 (_____).
- In R+1 cycle, send:
  - Assoc# and line state of line to replace (swap).

3. Move Out

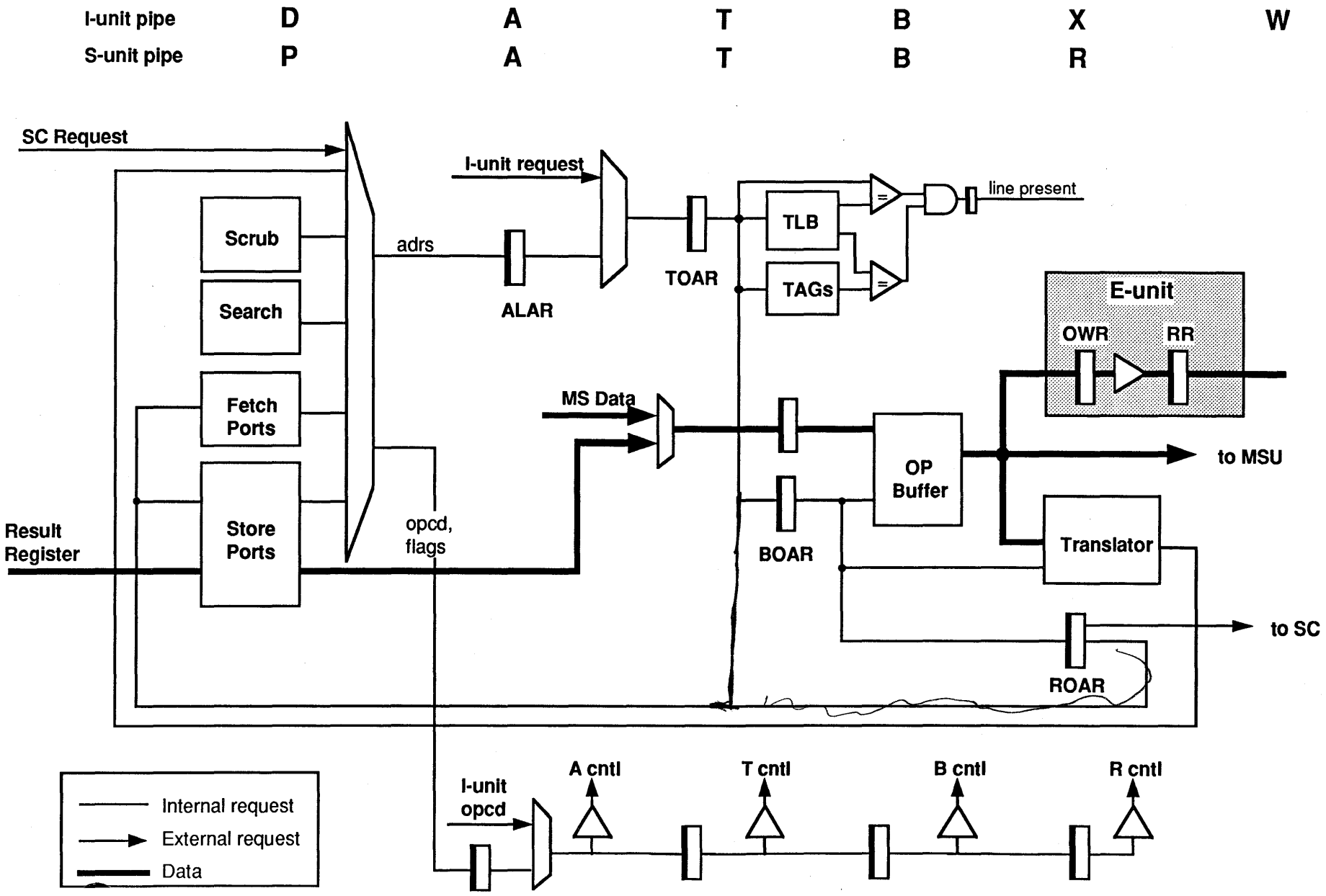
- Initiated by SC an indeterminate number of cycles later.
- Has three basic flavors, based on swap line state:
  - Short: line is public. Takes one flow to change it to invalid.
  - Private Short: line is private but unmodified. Takes 2 flows.
    - * First flow verifies it's still unmodified (if not, convert to LMO). Second flow invalidates.
  - Long (what's shown in the diagram): line is modified and needs to be backstored.
    - * Two flows needed to read out both half lines.
    - * Muxing to System Storage takes 4 cycles, so the flows are spaced 4 cycles apart.
- This is called a Swap MO.

4. Move In

- Initiated by the SC an indeterminate number of cycles later.
- Data transferred 1 QW per cycle, starting with the QW containing the requested data.
- LDBYPTAGAD flow
  - Loads BYPTAGAD register with the address of the data being moved in.
  - Generates a P-cycle Kick signal to the ports.
- Awakened by the Kick signal, the fetch port retries the fetch the next cycle.
  - BYPTAGAD is matched against the TLB, along with the TAGs.
  - If it matches (which it will in this case), the data is selected from the bypass path.
  - To save a cycle, data can be bypassed before going through ECC.
- MI1 and MI2 flows
  - After 4 QWs fill the data in register, MI1 flow writes them.
  - Similarly, MI2 flow writes the second 4 QWs.

S-unit Basic Blocks

Rev. 1, 5/91



Priority

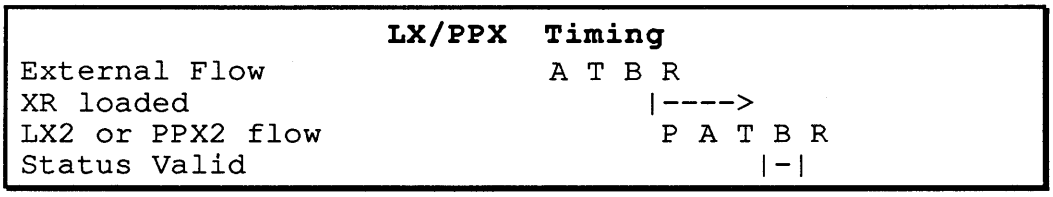
- **Overall S-unit priority structure:**

1. SC
2. Store Port Hi
3. Translator
4. Fetch Port
5. I-UNIT
6. Store Port Lo
7. Search Machine
8. Scrub

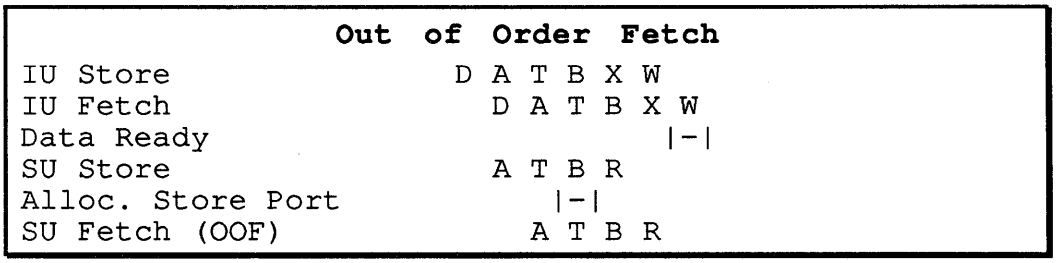
Sc
St Hi
Fetch
Translate
St. Lo.
I-Unit
search machine
scrub

Some Advanced Stuff

- **Line crossers (LX), potential page crossers (PPX)**
  - For LX, operand to be fetched spans 2 lines, requiring 2 buffer accesses. First access loads OWR, second access overclocks only those bytes that come from the 2nd line.
  - For PPX, future operands for the instruction *may* cross a page boundary (e.g. MVC). If the second page gets an exception, this needs to be determined early on in the alg.
  - I-unit sends flags indicating PPX and LX (could even be both).
  - XR Complex contains registers (1 for LX, 1 for PPX) that can be loaded with an incremented version of the BOAR (increment to next line or page, appropriately).
  - For the second flow, the appropriate XR register is selected into the P-cycle instead of the fetch port.



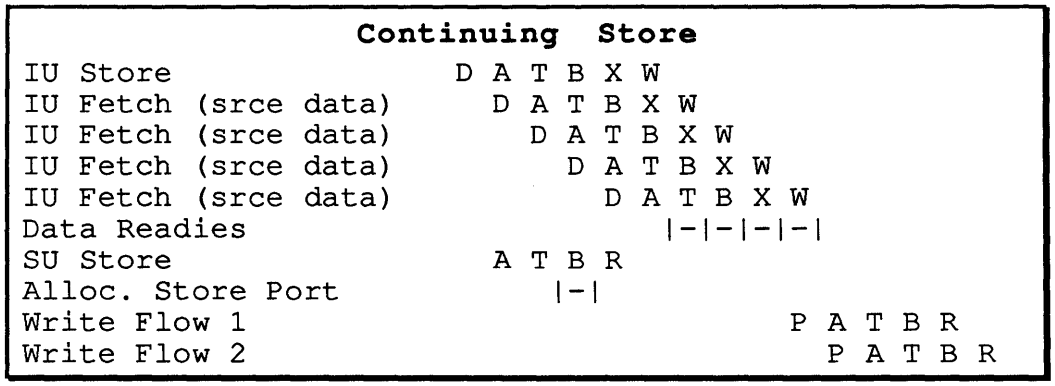
- **Out of Order Fetches (OOF)**
  - For SS OPs the address for the store comes from the 2nd HW of the instruction, and the address for the fetch comes from the 3rd HW. As a result it's more convenient to do the store flow first, followed by the fetch which will provide the data used by the store.
  - This fetch is called an Out of Order Fetch to the S-unit. The main difference is that the fetch can ignore SFI w.r.t. the store port containing the associated store. This associated store is logically after the fetch, so the fetch can (and must) proceed before the store completes, even if the addresses overlap.



Some Advanced Stuff (continued)

- **Continuing Stores**
  - Used when the I-unit wants to do a series of contiguous stores to the same line (e.g. Store Multiple).
  - Special Store Queue buffer can hold up to 64 bytes (8 DW), all associated with 1 port.
  - This buffer can write 16 bytes per cycle to the cache.
  - Thus, you need to do:
    * 1 store flow to allocate the port.
    * 1 write flow per QW.

    which is a lot faster than 2 flows (1 store, 1 write) *per DW*, as it would be otherwise.



- **Store Propagate**
  - Some SS ops (e.g. MVCL) can be used to store the same data value to all bytes of the destination field. This is called *propagation.*
  - When doing such a propagation, the Store Port only needs to be loaded with 1 DW of data. This doubleword can then be simultaneously written to multiple DWs in the buffer (up to 64 bytes) using just 1 write flow.

- **Line Store**
  - A special case of the above propagation is used by the operating system to do page clears - the same byte (typically 00) is propagated to an entire 4K page of data.
  - Often this page is cleared in anticipation of allocating it to a process. Since it isn't yet allocated, no further references will be made to it for a while, so you'd rather do the stores to MS without bringing the page into the cache and displacing more useful data.
  - Accordingly, the I-unit detects this case and generates a Linestore to the S-unit. The S-unit, in turn, passes the Linestore on to the System Controller, which will propagate the byte throughout a line of data directly in Main Store.

CPU Performance Analysis

$$\text{MIPS} = \frac{1000}{\text{P (ns/cyc)} \times \text{I (cyc/instr)}}$$

$$\text{I} = \text{E} + \text{D} + \text{S} + \text{M}$$

1.5 - 1.7

Execution

- nominal instruction execution time, assuming no interlocks.
- Function of:

.7 - .9

_____
_____
_____

Delay

- delays due to pipeline interlocks, other than FDI, including:
  - I-fetch: Branch penalties, other IF disruptions due to branches
  - Pipeline interlocks: EGI, OPI, etc.
- In addition to instruction mix and µcode, this is a function of:

1.3 → 2.5

_____
_____
_____
_____

Storage

- FDI delays - waiting for buffer data.

  $S = \mu_i \cdot M_i + \mu_o \cdot M_o + \mu_{tlb} \cdot M_{tlb}$ (µ = miss rate, M = miss penalty, i means IF, and o means OP)

- In addition to instruction mix, this is primarily a function of:

.03 - .10

_____
_____
_____

MP Serialization

- Some instructions require the CPUs to synch up (get between units of operation) before the instruction is executed. Each CPU completes the current unit of operation, then waits until the instruction is executed. Thus, the Initiating CPU pays a penalty waiting for the others, and the Receiving CPUs pay a penalty each time they have to stop and wait.

  M = Initiator Rate • Initiator Penalty
  + Receiver Rate • Receiver Penalty

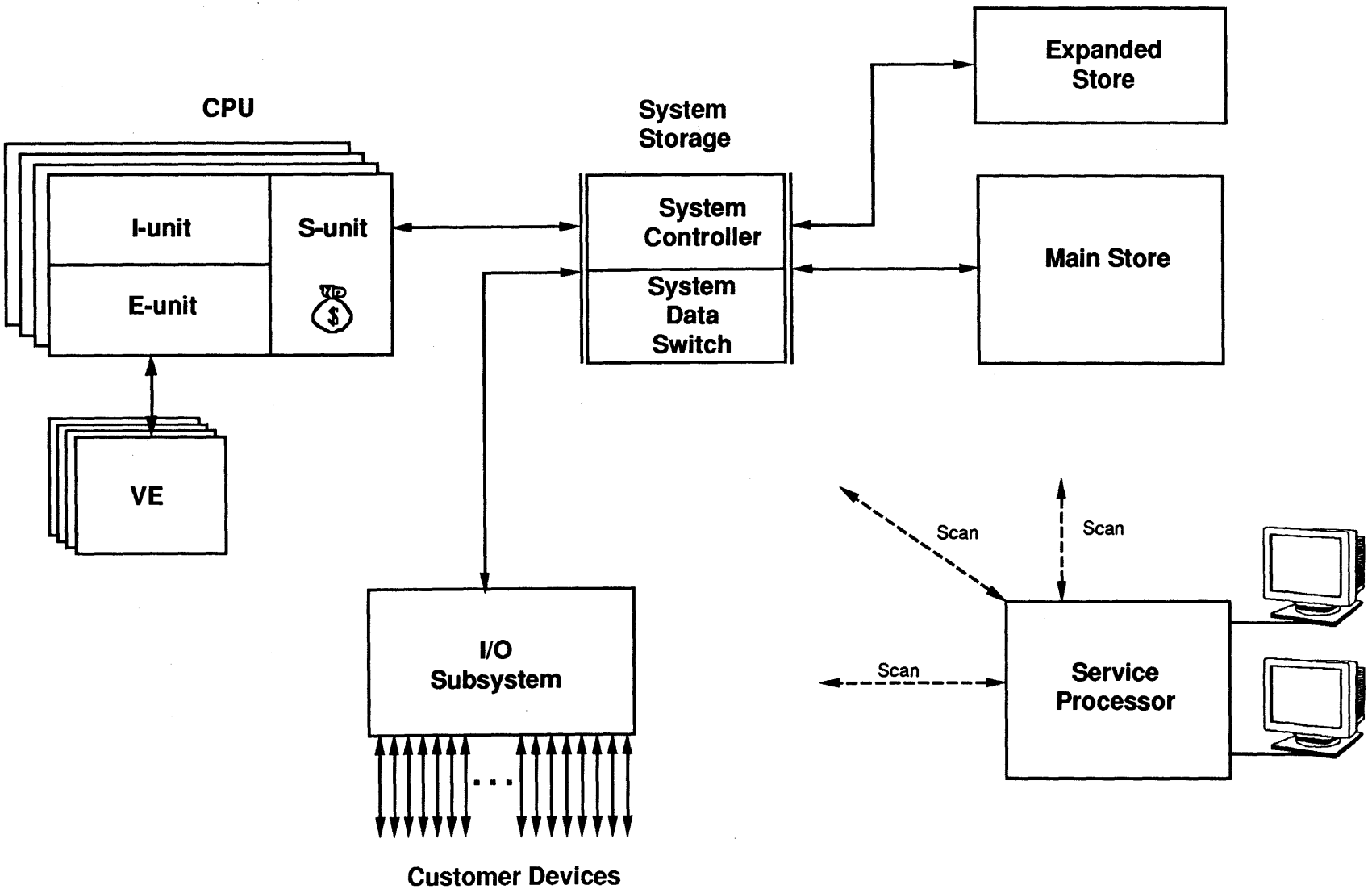
- In addition to instruction mix, M is primarily a function of:

_____
_____

System Storage

SONA Overview - SS System

Rev. 2, 8/91



SONA Overview

- **System Storage is the focal point for data transfer between:**
  - CPU(s)
  - IOP(s)
  - SVP
  - System Storage itself

- **System Storage includes:**
  - Main Store Array
  - Key Array
  - XSU Controller/Array
  - System Data Switch (provides connectivity between CPUs/IOPs and data)
  - System Controller (address and control focal point)

SC Opcode List
(condensed)

SU MS Data Ops

→ **FETCH - 4 flavors (Public/Private x fetch/prefetch)**
DECLARE PRIVATE
→ **S-UNIT LMO**
LINESTORE - 4 flavors

COPY REASSIGN OPCODES (2)
RELEASE CACHE LINE

SU MP & Key Ops

PURGE - 7 flavors
SSKNP Set Storage Key Non Propagate
SCRB Set Reference and Change Bit
RRB Reset Reference Bit
ISK I-Unit Key Fetch
TLB KEY REQUEST (S-Unit Key Fetch)
SSK Set Storage Key Propagate
SWK Swap Storage Key
IPTE - 13 flavors
LDMRUMSGprop PROPAGATE LOAD MRUT

XSU Ops

PGOUT MS ADRS Page-out Mainstore Addr
PGOUT XS ADRS Page-out Extended Storage Addr
PGIN MS ADRS Page-in Mainstore Addr
PGIN XS A
~21 other XSU Ops

IOP Ops

FETCH - 5 flavors
RELEASE LOCK
STORE 5 flavors

Internal MS Ops

→ **MAINSTORE WRITE**
MAINSTORE SCRUB

SC Opcodes

- **All requests to System Storage come through the SC.**

- **SC Design is oriented around the Data Ops, esp. Fetch.**
  - A Fetch request from the S-unit leads to a _____.
  - May say Public is OK, or may ask for it Private.

- **LMO**
  - the SC initiates this by sending LMO pipeflows down the S-unit pipe.
  - These flows return the address and data to System Storage in the form of a LMO "request" to the SC.

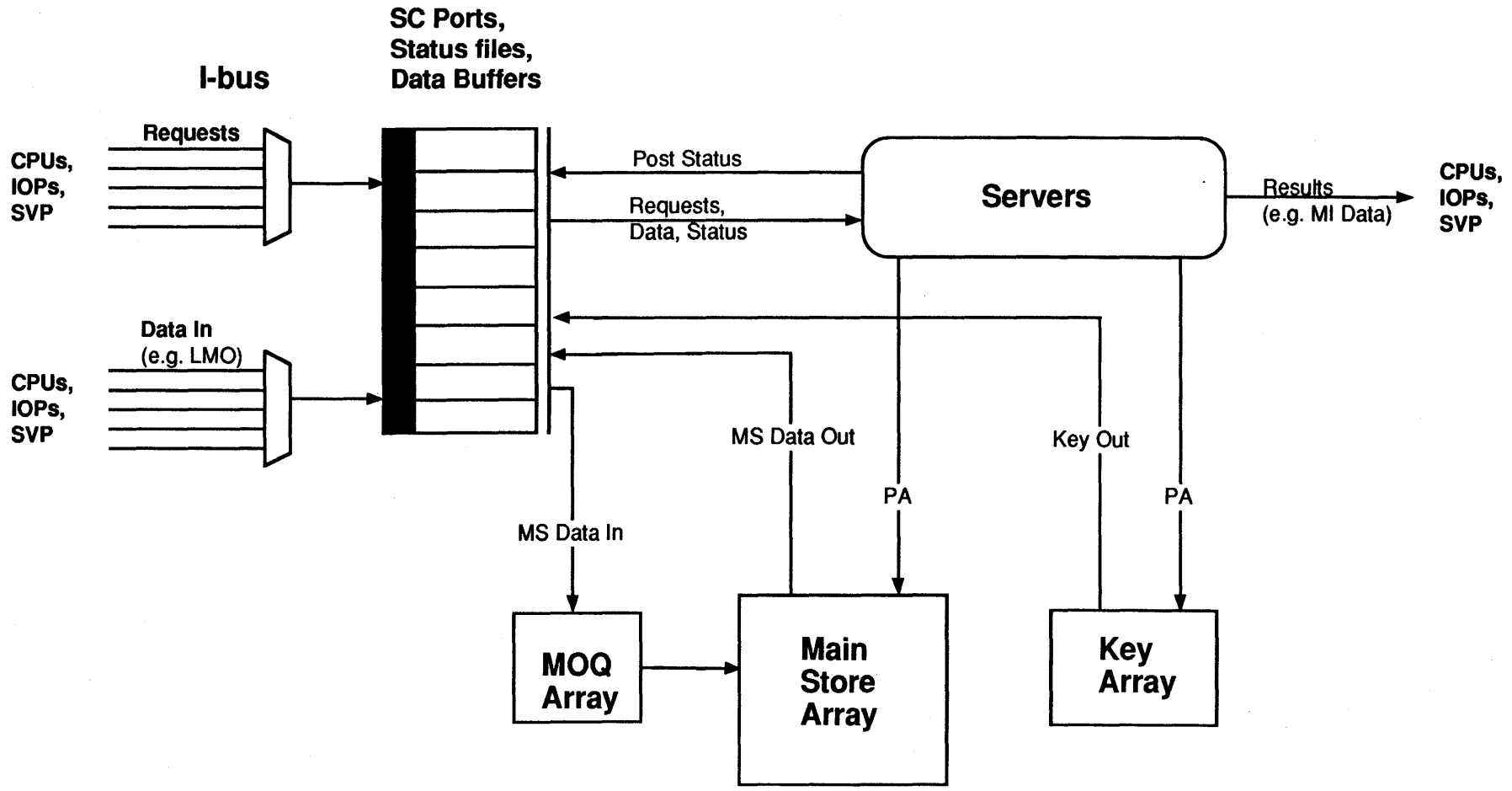
- **MP and Key Ops**
  - MP Ops involve propagation of the operation to other CPUs.
  - Key Ops operate on the Key Array.

- **XSU Ops**
  - Data transfers between the XSU and the MSU.

- **IOP OPs**
  - Fetches are similar to S-unit OPs.
  - Lacking a cache, the IOP does Stores directly to the MSU. Similar to a LMO.
  - To do Read-Modify-Write, the IOP can lock a line. The SC maintains this lock.

- **MS Write**
  - The actual writing of data to the Main Store.
  - Done in the background, thanks to the Move Out Queue.

System Storage Basic Concept Rev. 1, 8/91



Basic System Storage Concepts

- **The I-bus chooses the highest priority request and loads it into the SC ports**
  - A request includes a packet with enough information to process the request.
  - If the I-bus doesn't accept a request, it's up to the requestor to try again.

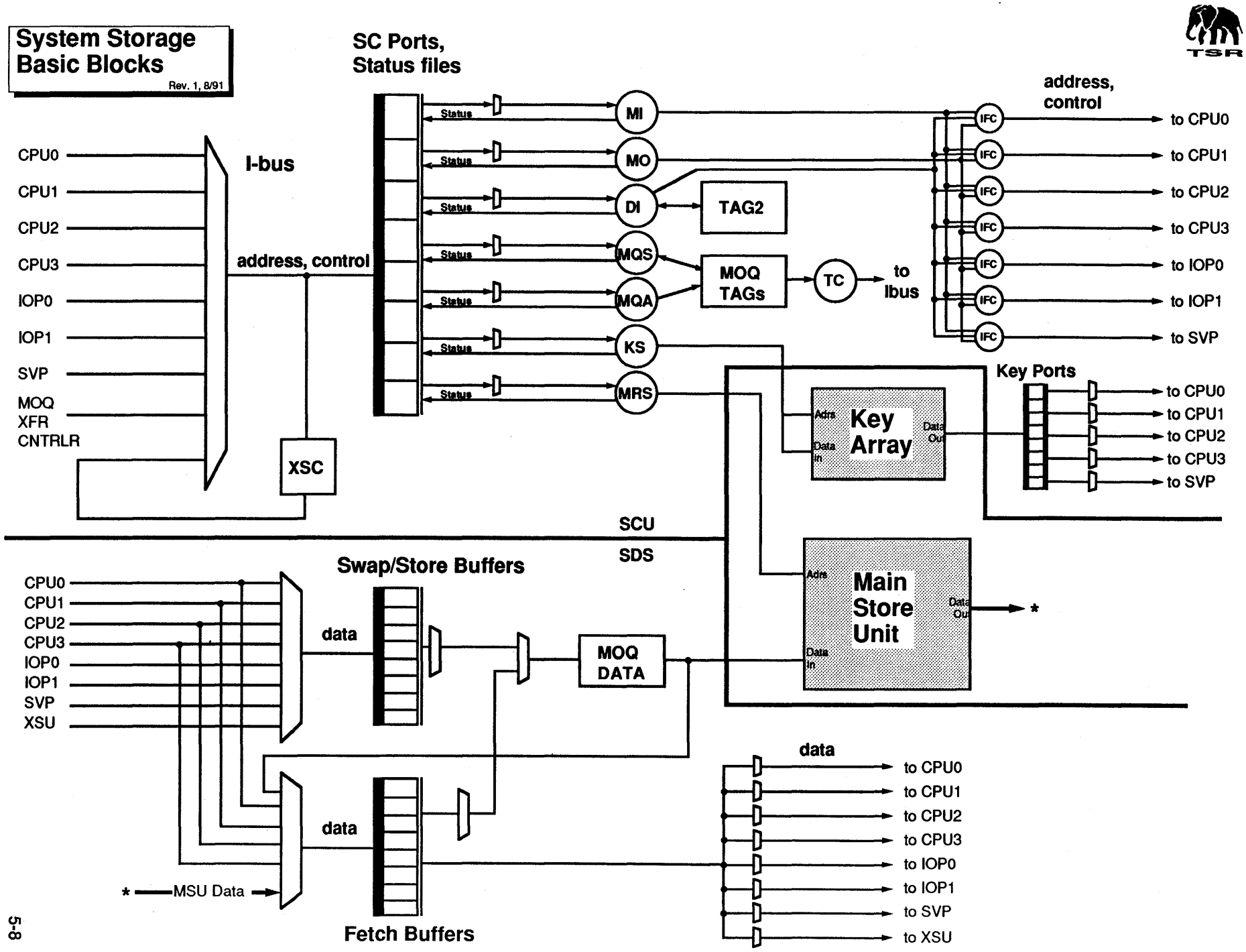
- **The SC Ports are the focal point of System Storage**
  - Central mailbox containing everything dealing with a request, including:
    * The initial request.
    * Current status of processing.
    * Any data associated with the request.
  - FIFO Queue: I-bus loads Bottom of Queue Port, servers process Top of Queue.

- **Servers provide the control to process the request**
  - Send addresses and control to the arrays.
  - Transfer results to the requestor.
  - Process requests independently. Communicate with each other through status bits.
  - Each server proceeds at its own pace. Each server has its own TOQ.

- **Arrays provide storage for Data and Keys**

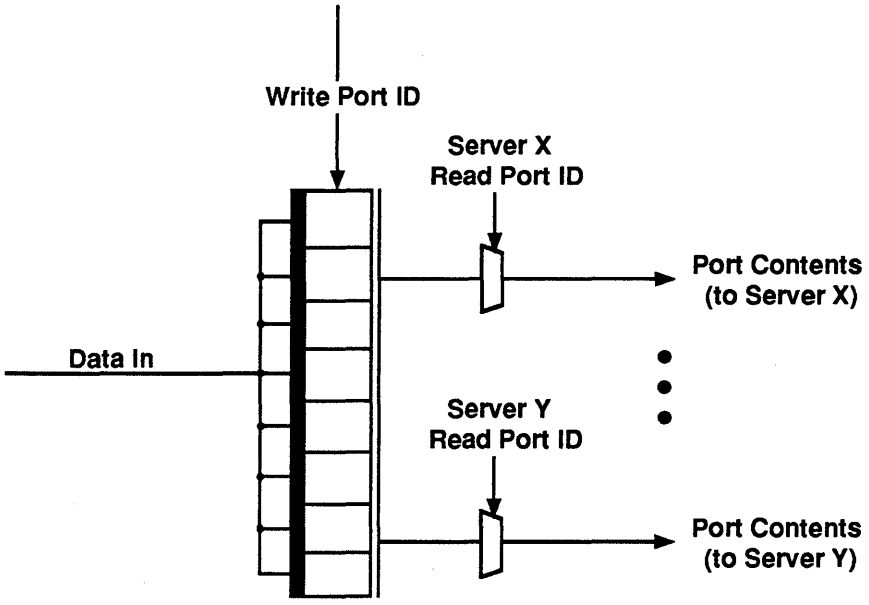
- **Basic actions needed to complete an SU Fetch:**
  - _____
  - _____
  - _____
  - _____
  - _____
  - _____

- **Move Out Queue provides buffering for writes to Main Store**
  - Holds Move Out data while Main Store is busy doing the read.
  - Maintains data in a queue, does write to MS in background.
  - Analogous to _____



System Storage Basic Blocks

- **I-bus**
  - Selects request to load into the ports.
- **SC Ports**
  - Provide storage for the initial request.
  - Separate output selectors for each server, allowing servers to go at their own pace.
  - Also provide individually writeable status bits allowing each server to post its status.
  - 8 ports in an SS system, addressed by Port ID.
- **Data Buffers**
  - Conceptually an extension of the SC Ports, but often referred to separately.
  - a.k.a. Port Data Buffers (PDBs).
  - Swap/Store Buffers hold Swap LMO data.
  - Fetch Buffers hold MI data.
- **Key Ports**
  - Conceptually an extension of the SC Ports.
  - Hold data read out of Key Array.
- **Arrays**
  - Main Store, Key, and Move Out Queue.
  - MS and Keys implemented on BLCs, MOQ is in SIMTEC.
- **Servers**
  - Each server has its own Port ID to read out a request from the SC Ports.
  - **Main store Request Server:** sends address and control to the MSU.
  - **Key Server:** sends address, control, and data to the Key Array.
  - **Data Integrity:** searches all caches for data, initiates DI Move Out if needed.
  - **MO Server:** initiates Swap MO based on replacement info from S-unit.
  - **MI Server:** initiates MI flows to S-unit and controls data transfer out of Fetch Buffers, based on status posted by ports. In general, wraps things up for a request.
  - **MOQ Search Server:** searches MOQ for data.
  - **MOQ Add Server:** transfers MO data from Ports/Buffers into MOQ.
  - **MOQ Transfer Controller:** initiates transfer of data from MOQ to MS, via SC ports. Not a "server" as it doesn't process SC Port requests.
- **Interface Controllers**
  - Provide actual control and address interface to the S-unit pipeline.
  - MI, MO, and DI (path not shown) may all contend for a given IFC.



Port Structure

- **Looks like a multi-ported RAM.**
  - Has data in, write address (Write Port ID).
  - Multiple read paths provided, each with a separate read address (Read Port ID).
  - Read paths customized: only provided for servers that need them.
    * A given server may have several selectors covering different bits. This allows the server to read different bits at different times.
  - e.g. SC Ports
    * Data is the original request.
    * Write Port ID is _____ .
    * Read paths for every server.

- **Each port includes multiple pieces which are all variations of this structure.**
  - SC Ports (original request)
  - Swap Address Buffers
    * The PA of the line to be swapped out on a fetch is sent over much later than the initial request and is stored in a special section of the port called a Swap Address Buffer.
  - Various status bits
  - Data buffers (Swap and Fetch data)
  - Key Ports

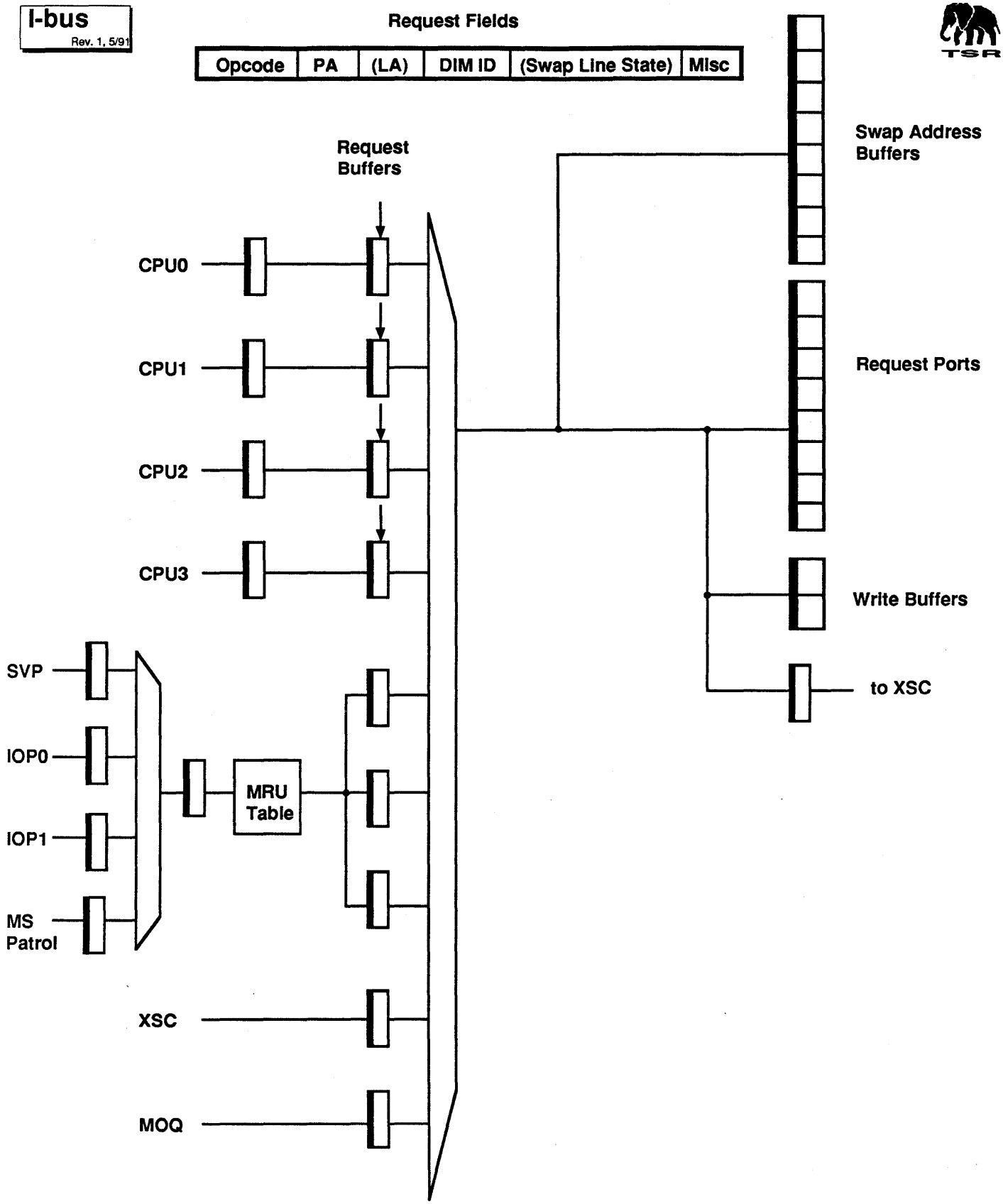
- **Each server processes like a FIFO queue**
  - Different servers may be on different requests at the same point in time, but each server cycles through the Ports on a FIFO basis.

- **Status Files**
  - Each server sets one or more status bits, including:
    * Done bits (1 per server) indicating server is done with request. Stays set until port is overclocked with a new request.
    * Timing bits: Provide timing information to other servers.
    * Results: Specific results obtained by the server.
  - Separate Write Port ID is provided for each server's set of status bits.
  - Read paths customized for each bit

TSR

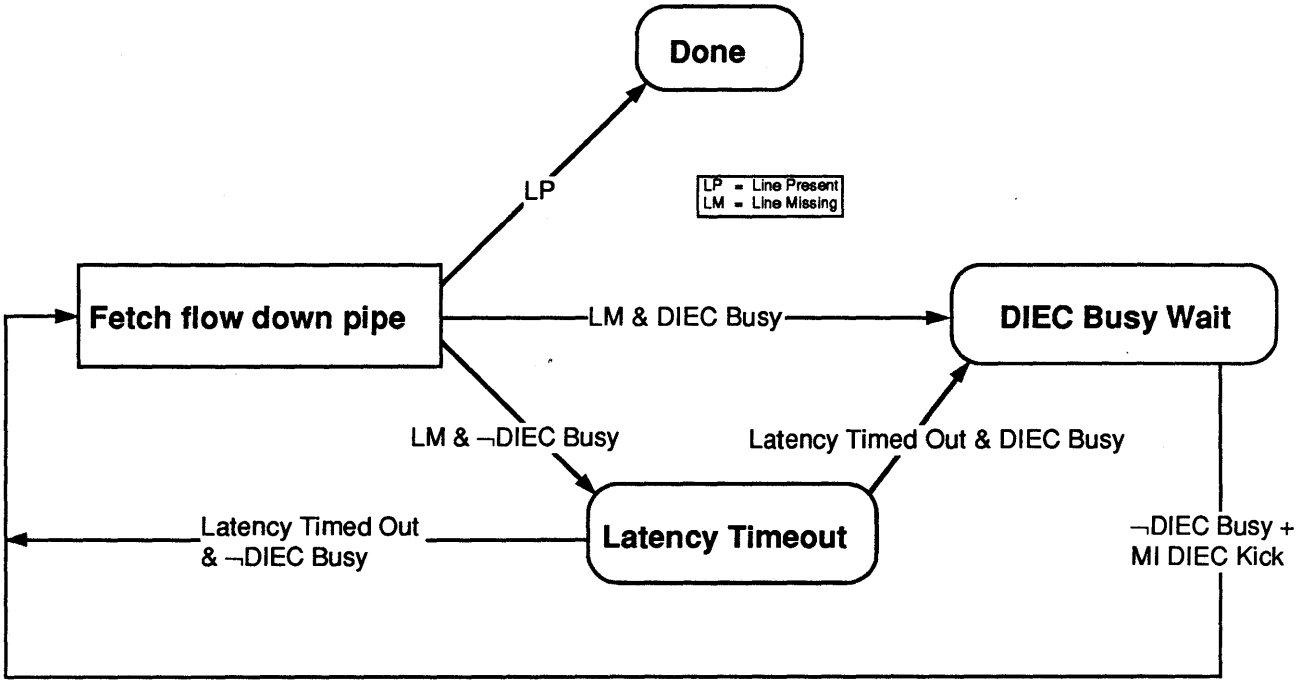
Request Fields

Opcode	PA	(LA)	DIM ID	(Swap Line State)	Misc



I-bus

- **Highest priority request (assuming no busies) accepted into I-bus. Includes:**
  - Opcode
  - Physical Address
  - Logical (Effective) Address (S-unit only)
  - Dimension ID
  - Swap Line State (S-unit only)
  - Miscellaneous stuff
- **Priority tree:**
  1. Long Move Out
  2. MOQ HI
  3. MS Patrol
  4. eXpanded Storage Controller
  5. SVP
  6. IOP - ties broken by toggle latch
  7. S-unit (non-LMO) - ties broken by toggle latches
  8. MOQ LO
- **Busies used to protect resources**
  - MS Element Busy
    * Based on PA0:2
    * Protects MS RAMs from a second access while first is still busy.
  - DIEC Busy
    * Based on PA21:24
    * Prevents multiple requests to same line from being in SC at the same time.
    * Stands for Data Integrity Equivalency Class.
    * Pronounced DEEK.
  - If the winning request has a conflict with a busy, it isn't validated in the SC ports.
- **Most requests go from I-bus into SC ports. Exceptions include:**
  - Swap LMO: goes into Swap Address Buffer for originating port.
  - MS Write requests (from MOQ only) go through Write Buffers.
  - XSC requests go through a dedicated XSC port.
- **Request buffers hold pending CPU requests until they're accepted.**
  - a.k.a. Holding Registers.
- **IOP and SVP addresses may be absolute, requiring MRU Table access.**
- **NOTE: "S-unit" and "CPU" used interchangeably.**



Nominal MI Timing

```
Fetch Flow      P A T B R
Tag Miss             |-|
Send MI Req            |-|
Latency Count           5 4 3 2 1 0
DIEC Busy                          |---------->
-------------------------------------------------

LdBypTagAddr                                    P A T B R
MI DIEC Kick                                   |-|
Fetch Flow                                       P A T B R
```

5-13.1

SC-SU Interface - DIEC Busies

SC Sends a copy of DIEC Busies to the SU

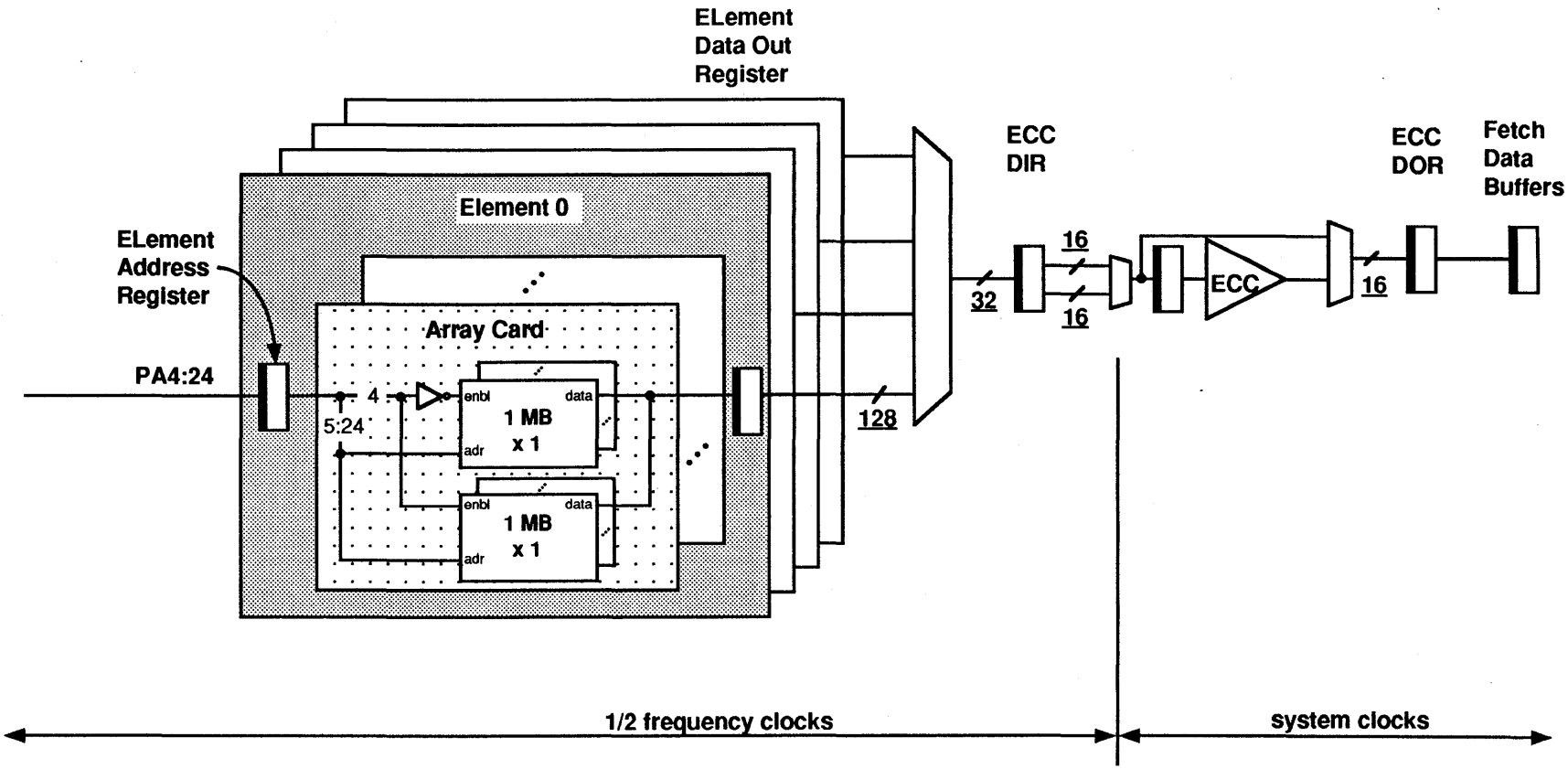
- these are used to kick fetch ports out of DIEC Busy Wait states.
- also used in the B-cycle to determine whether or not to send a request to the SC.

Fetch Flow gets Line Missing:

- If DIEC Busy is already on (in the B-cycle) no request is sent and the flow goes directly into a DIEC Busy Wait State.
- If DIEC Busy isn't on then the request has a chance to get into the SC:
  1. Send a request to the SC.
  2. Wait a while so the DIEC Busy has time to come on.
     NOTE: implemented by going into DIEC Busy Wait and forcing DIEC Busy with a counter.
  3A. If it doesn't come on then assume the request failed and recycle.
     NOTE: it could succeed out of _____ during the recycle.
  3B. If it does come on, then assume the request succeeded and wait in DIEC Busy.
     NOTE: it could've actually failed and the DIEC Busy is due to a different request.
- Recycle when the DIEC goes available, or when kicked by the LdBypTagAddr flow. This kick is DIEC specific.

MSU Data Paths

Rev. 1, 8/91



MS Addressing

Physical Address	0	1:2	3	4	5	24
1Mx1 SRAMs	Side Select (DS)	Elmnt ID	Unused	Bank ID	RAM Address	
4Mx1 SRAMs	Side Select (DS)	Elmnt ID		RAM Address		

MSU Data Out Paths

- **1 MB RAMs dotted in pairs to create 2Mx1 structure**
  - PA5:24 addresses the RAMs
  - PA4 selects RAM to enable

- **2M x 128 byte lines per element**
  - 128 RAMs (64 pairs) per array card = 2M x 64 bits per card
  - 20 array cards per element = 2M x 1280 bits = 2M x 128 bytes + ECC
  - Can read or write an entire line at a time

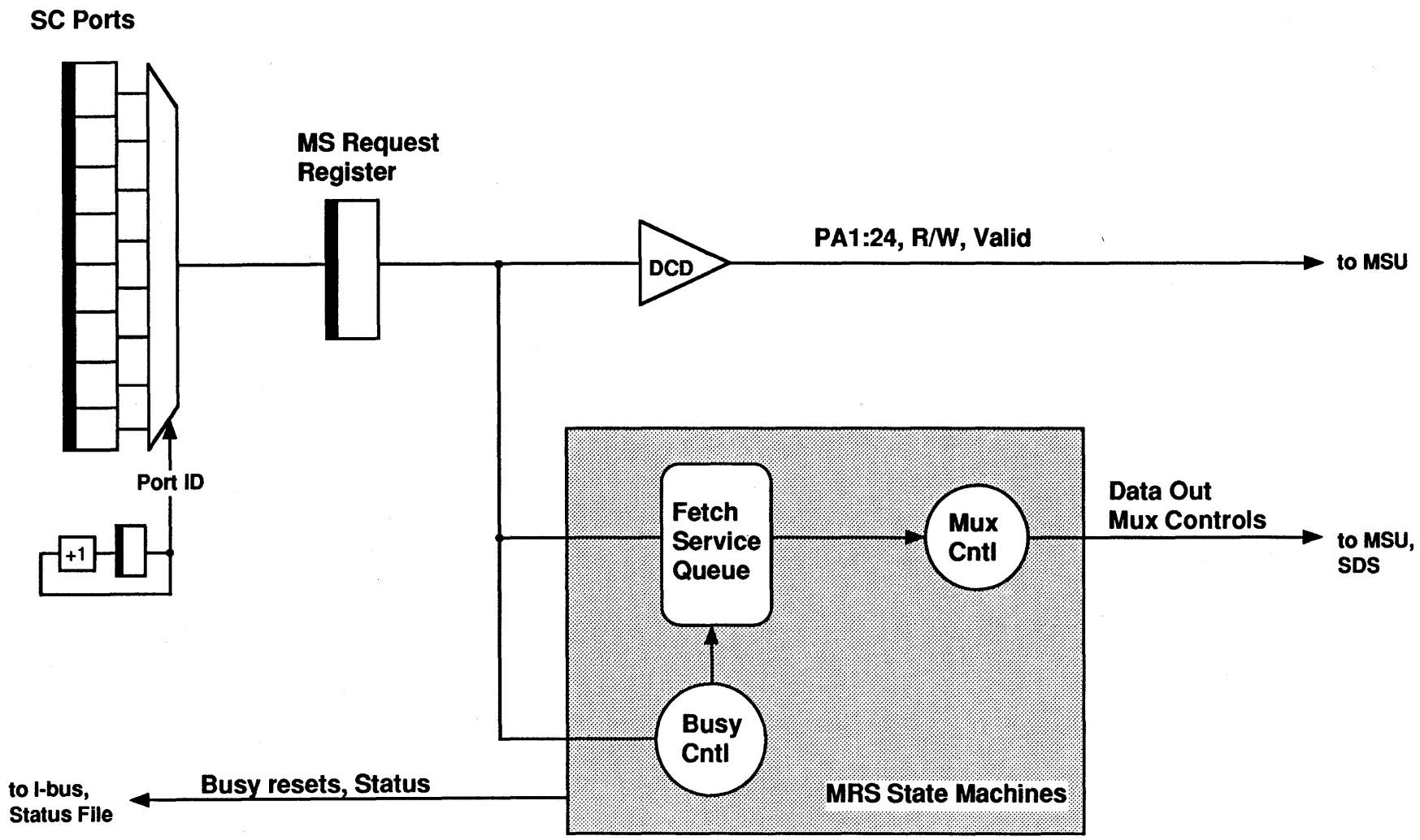
- **4 elements per side**
  - 4 x 2M x 128 = 1 GB/side

- **Data Out MUX (16 to 1) selects source element and muxes quarter lines (32 B) into ECCDIR.**

- **ECC speed matches to load Fetch Data Buffers**
  - MSU runs at 1/2 speed clocks, SDS is on full speed clocks.
  - ECC (on SDS) selects 16 bytes/cycle from 32 byte ECCDI register.

Main Store Request Server

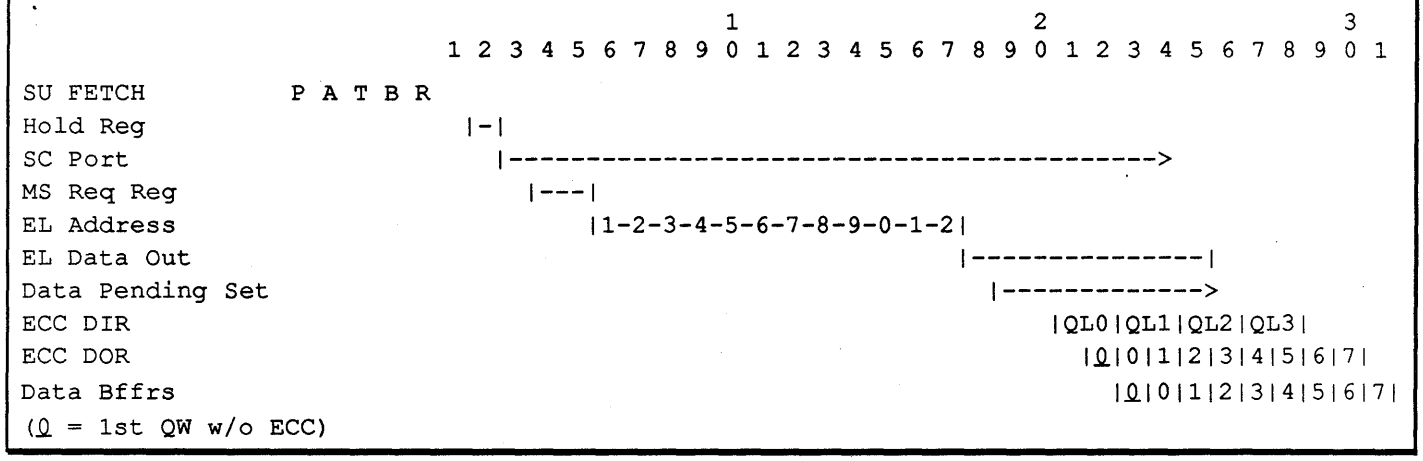
Rev. 1, 8/91



MS Request Server

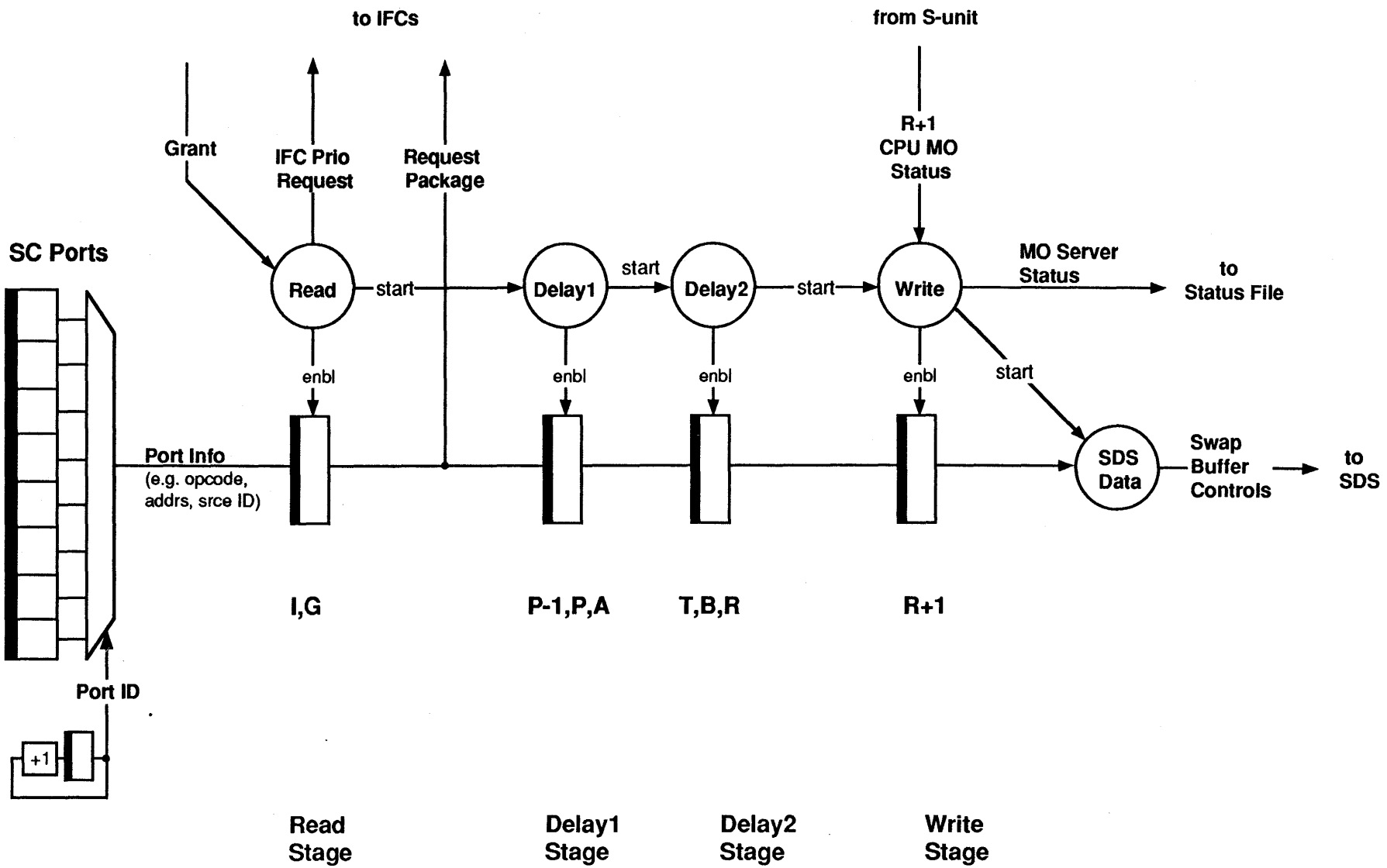
- **Next active request loaded into MS Request Register.**
- **Request sent on to MSU**
  - PA1:24
  - Port opcode decoded to 1 bit, plus a Valid bit.
    * 1 bit indicates Read or Write.
- **State machines informed of the request**
  - Busy Control tracks timing of busies for I-bus.
  - Fetch Service Queue:
    * Tracks fetch requests that have been sent out.
    * Initiates Muxing out when data ready.
  - Mux Control controls Data Out Mux and ECC on MSU and SDS.
- **Status bits posted, as appropriate, to inform other servers of progress.**

Sample MRS Timing
(SU Fetch, 25 ns RAMs)



Move Out Server

Rev. 1, 8/91



MO Server

- **Based on Replacement Line state, initiates Swap MO.**

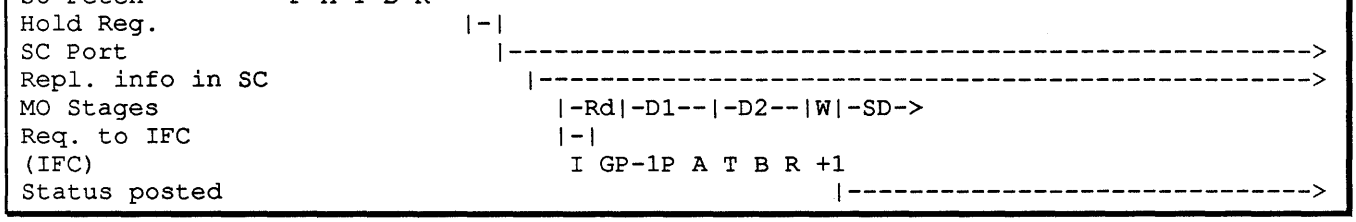
- **5 state machines pipelined together:**
  - Read State Machine:
    * Analyzes request to determine if a Swap MO is needed.
    * If so, requests IFC priority.
    * When given IFC grant, passes control to Delay1 state machine.

  - Delay1,2 State Machines:
    * Each machine counts 3 cycles, then passes the request on to the next stage.

  - Write State Machine
    * Monitors MO status for line locked or other problems.
    * If LMO, initiates Data control state machine.
    * Posts status.

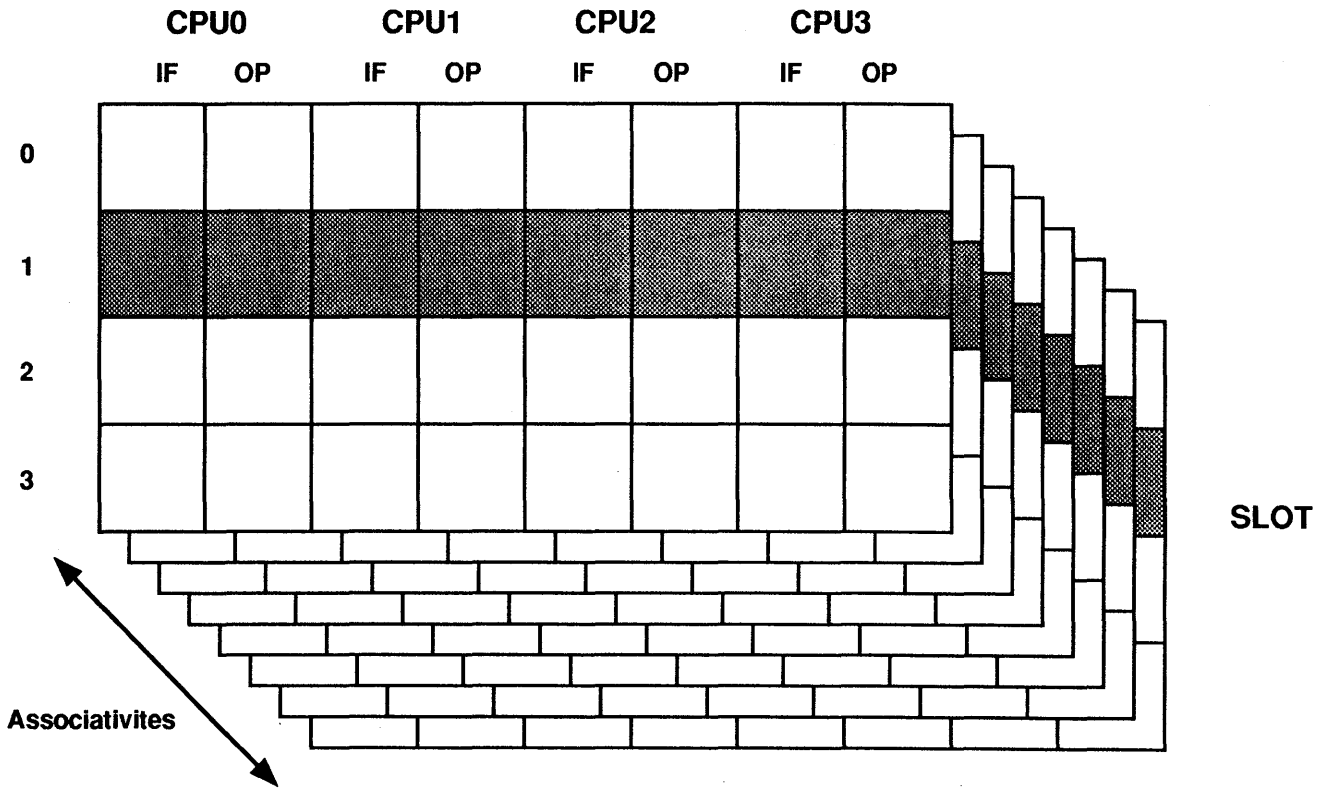
  - SDS Data State Machine
    * controls transfer of data into Swap/Store Buffers.

MO Timing Example



QP Cache Search Possibilities

Rev. 1, 8/91



QP Cache Search Possibilities

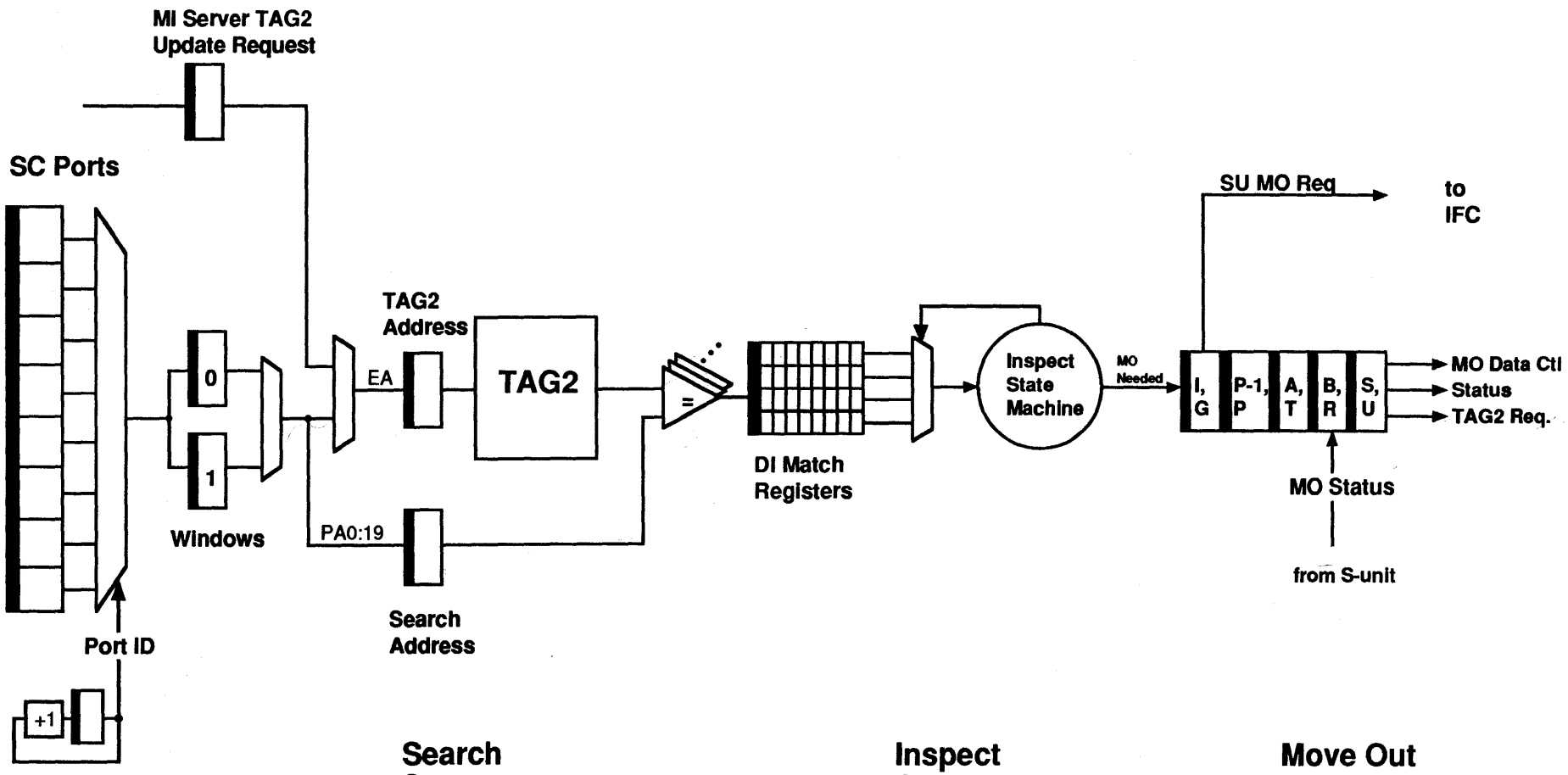
- **In a QP system, the requested data could be in**

  4
  x 8
  x 4
  x 2
  = 256

- **Each value of EA18:19 is referred to as a SLOT in the DI Server**
  - at most there are ____ matches per slot.
  - ___ total matches possible.

Data Integrity Server

Rev. 1, 8/91



P M R

DI Server

- **Responsible for:**
  - Finding any cache copies of requested line.
  - Initiating Move Outs, as appropriate, to ensure all caches follow the DI rules.
  - Bypassing data into Fetch Data Buffers for DI LMOs.

- **Focal point is TAG2**
  - Copy of all S-unit TAGs of a QP (or of 1 side of a DS system).
  - Each entry includes a valid bit, pub/priv bit, and PA0:19.
  - Organization allows 1 slot (i.e. EA18:19 value) to be accessed at a time.
  - Accessed via pipeline.
    * **P**riority
    * **M**atch
    * **R**esults
  - MI server can access to update during a Move In.

- **TOQ request loaded into available window**
  - Window holds request info for duration of DI processing.
  - 2 windows.
  - Saves long path of going out to SC ports and back to read info.

- **Search stage initiates 4 flows to search all EA18:19 values**

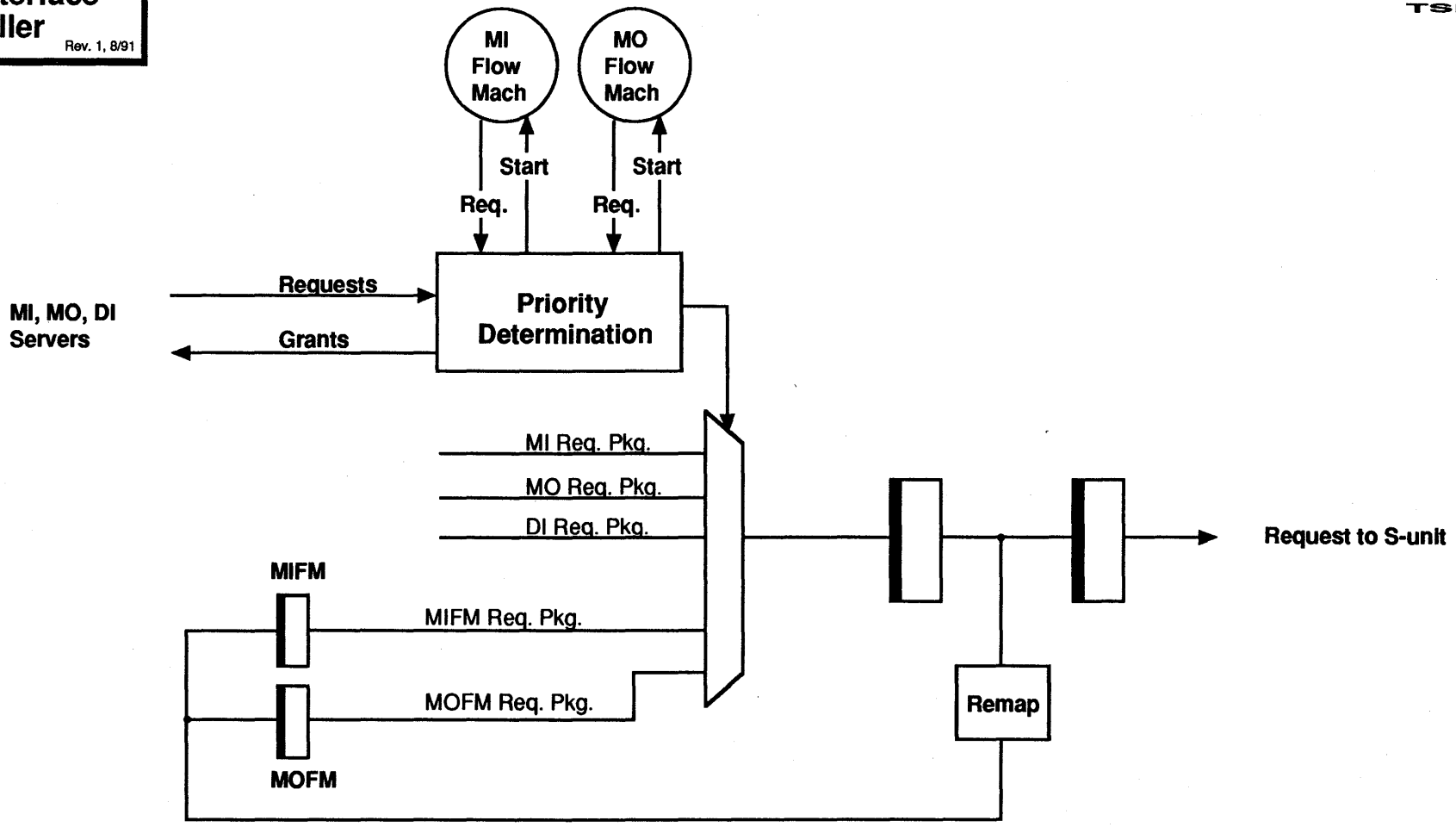
```
Slot 0      P M R
Slot 1        P M R
Slot 2          P M R
Slot 3            P M R
```

- **Inspect stage analyzes match results**
  - Results stored in DI Match registers.
  - 8 registers per slot (since only 1 associativity per cache set can match).
  - Inspect stage MUXes out 1 slot of the DIMRs at a time for analysis.
  - If MO's needed, MO stage is informed.

- **Move Out stage**
  - Requests priority to IFC, which in turn will control interface to S-unit.
  - Monitors status from S-unit to see if line is locked (or modified).
  - Controls data transfer into FDBs, posts status, and requests TAG2 access to update .
  - Addresses (for LMO and TAG2 update) come from window.

CPU Interface Controller

Rev. 1, 8/91



I G P-1

Request Package

SU Opcode	PA	EA18:19	Assoc. #	Flags, Misc.

Interface Controllers

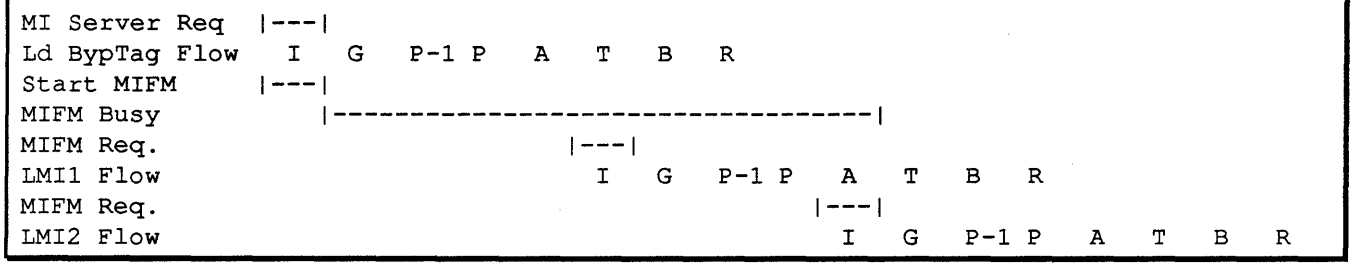
- **Responsible for controlling the interface into S-unit pipeline.**
  - Selects between DI, MO, and MI servers for priority.
  - Generates subsequent flows of multiple flow algs.
  - 1 IFC per S-unit.

- **Pipelined.**
  - **I** IFC priority
  - **G** Grant
  - **P-1** send request to SU (in P-1 cycle of S-unit pipe)

- **DI, MO, and MI servers contend for priority into IFC in the I-cycle.**
  - Highest priority request gets its request package sent down the pipe.
  - If this request is for a multiple flow algorithm:
    * Flow Machines (one each for MI and MO) are fired up to generate the follow on flows.
    * These follow on flows are highest priority.
    * The opcode and low order address bits are modified to form the follow on flows, and saved until needed.

- **Request package selected and sent to S-unit.**
  - Opcode
  - PA (exact bits depend on operation)
  - EA18:19
  - Associativity #
  - Flags, Misc.

Sample IFC Timing - Move In

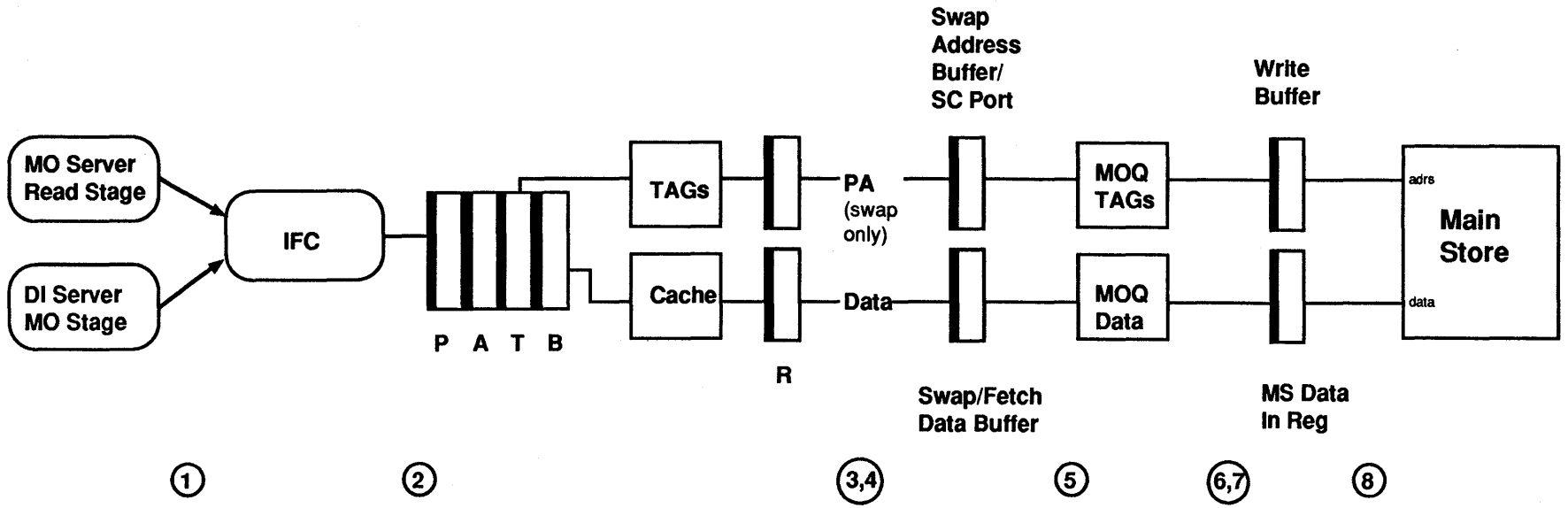


Long Move Out Process Flow

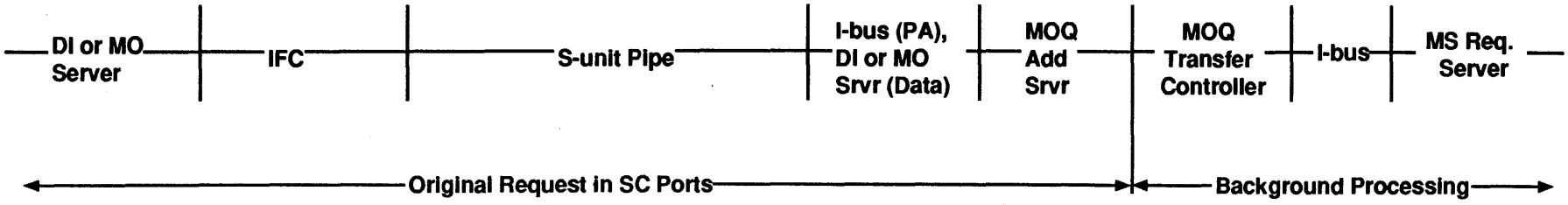
Rev. 1, 8/91

TSR

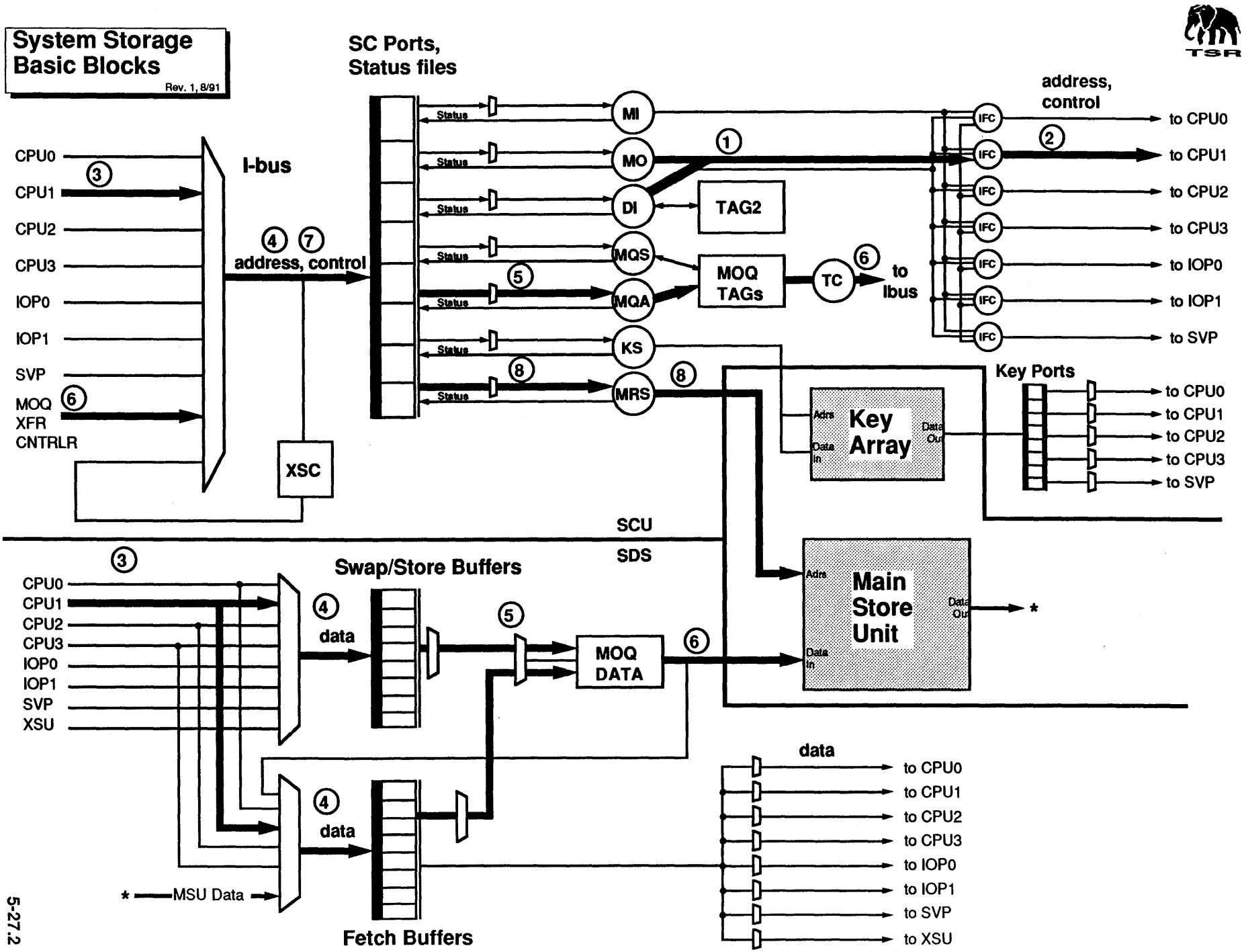
Address, Data Flow



Control Flow



- This page intentionally left blank -



Long Move Out Process Flow

1. **Initiated by MO** (Swap) **or DI** (DI LMO) **Servers.**

2. **Interface controller sends ___ flows down S-unit pipe.**

3. **S-unit sends out data from cache and PA from TAGs**
   - also sends Valid bit (line may be a Ghost).
   - also sends modified bit (on DI MOs, TAG2 only knows pub/priv. If line is unmodified, don't write MOQ).

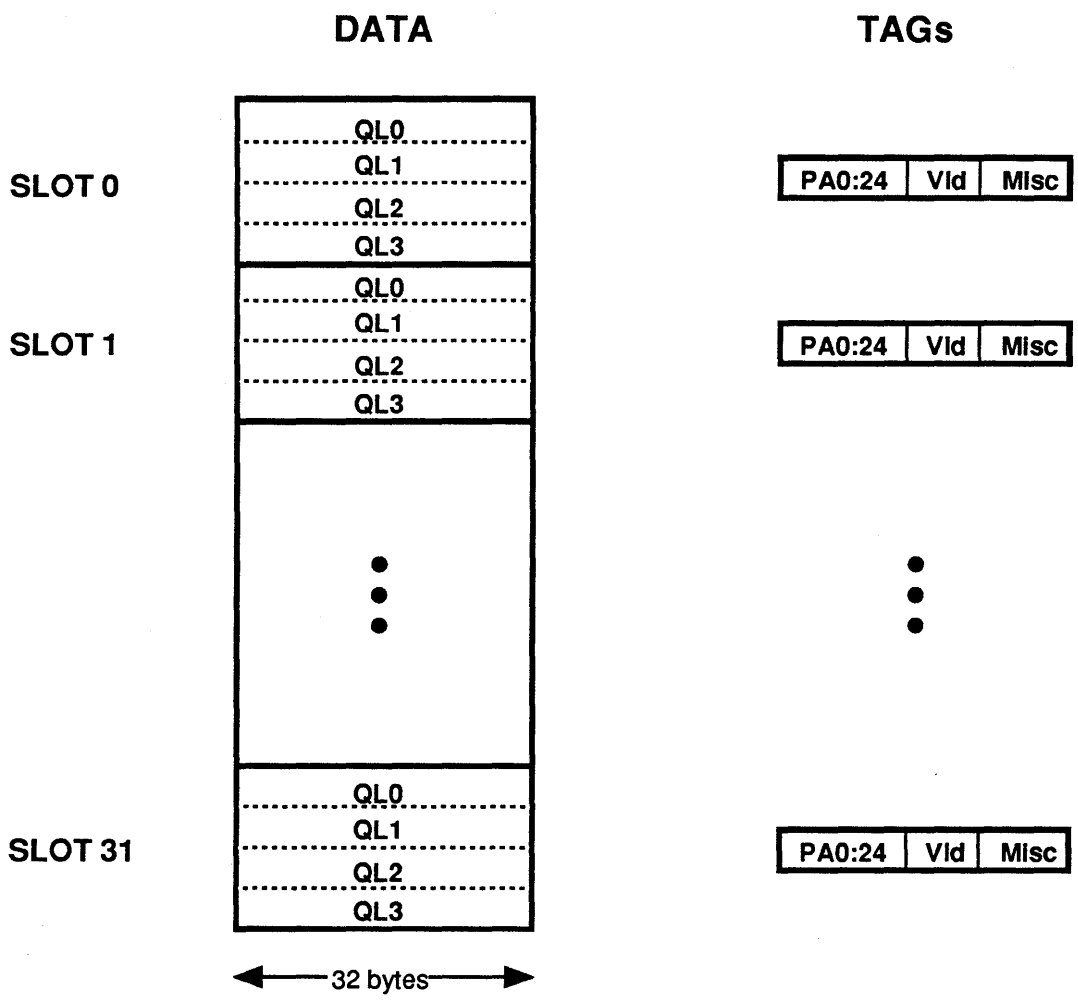
4. **The MO is loaded into the Ports.**
   - The PA is sent over the I-bus into the Swap Address Buffer.
     * PA only needed on Swap LMO. On DI LMO it's _____.
   - The data is loaded into a Data Buffer.
     * _____ for DI LMO.
     * Swap/Store Data Buffer for Swap LMO.

5. **MOQ Add server then puts the Move Out into the MOQ.**
   - DI or MO server post status bits telling Add Server the MO is there.
   - The Add Server loads the PA into a MOQ TAG.
   - The Add Server loads data into the MOQ array.
   - This is the end of foreground processing (i.e. original SC Port request is now done).

6. **MOQ Transfer Controller cycles through MOQ emptying out pending requests.**
   - Loads data into MS Data In Register.
   - Reads PA out of TAG and sends to I-bus as a MS Write Request.

7. **I-bus loads this request into a Write Buffer** (instead of using an SC Port).

8. **The MS Server generates a write request to the MSU.**
   - It sends the PA, plus an opcode saying to do a write.
   - The data is already set up in the MSDIR.



MOQ Organization

- **32 deep FIFO queue, 1 line per slot.**
  - MOQ can hold 4K of data.

- **Data Organization**
  - Can Read or Write 32 bytes (1 QL) per cycle.
  - 4 QLs per Slot.
  - Address includes 5 bit Slot Number and 2 bit QL number.
  - Implemented in RAMs.

- **TAG Organization**
  - 1 TAG per slot.
  - Includes:
    * PA0:24
    * Valid bit
    * Misc. bits
  - Implemented in latches.

- **Data and TAG Pipelines are accessed independently.**

- **Non-pipeline control provided by:**

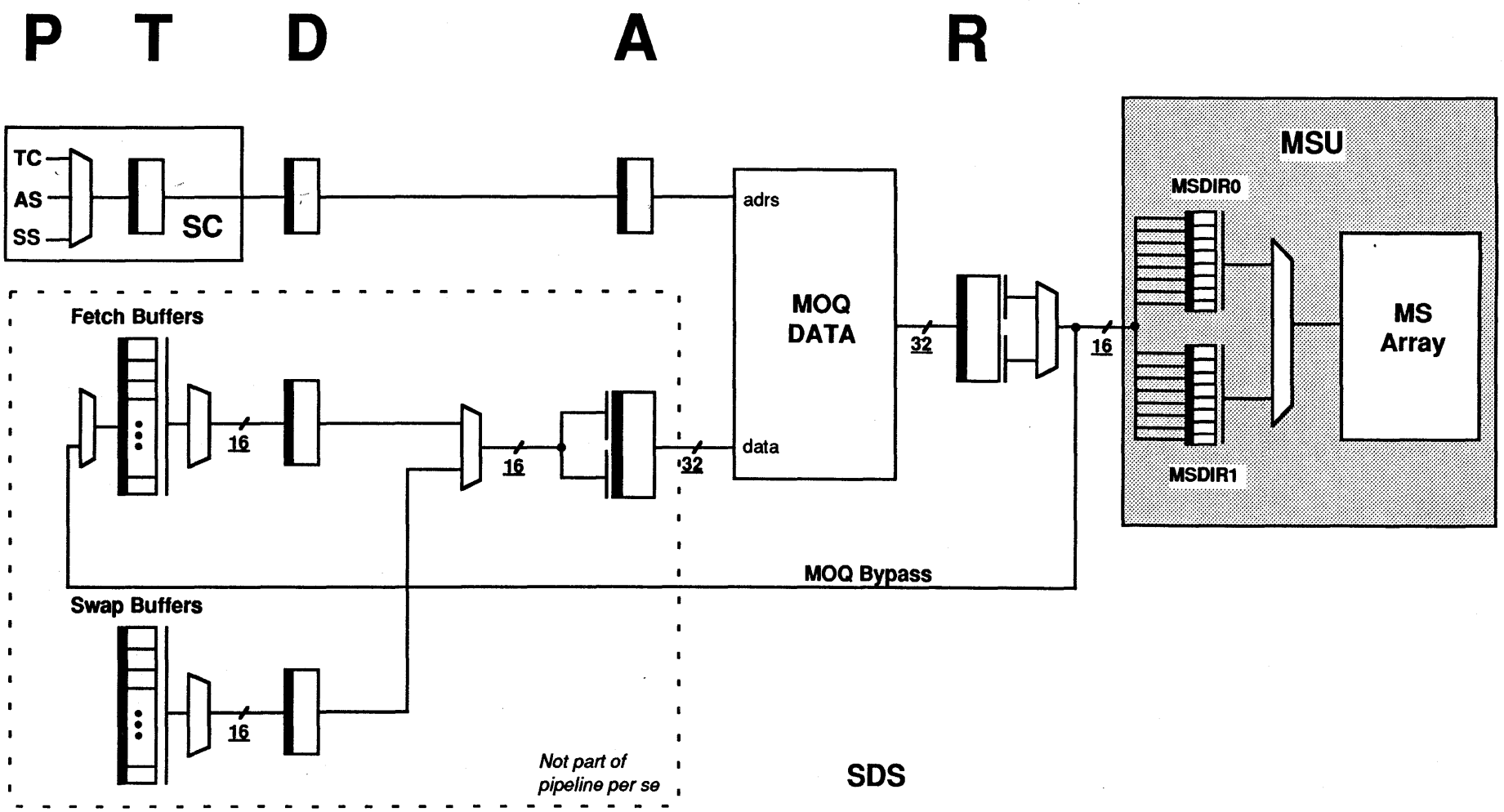
MOQ Add Server
- Writes data/address into MOQ.

MOQ Search Server
- For Fetch requests, searches MOQ to see if data is there.

Transfer Controller
- Transfers data out of MOQ and sends Write request to MS.

MOQ Data Pipeline

Rev. 1, 8/91



MOQ Data Pipeline

- **Priority determination and address selection**
  - Add Server contends to _____
  - Transfer Controller contends to _____
  - Search Server contends to _____

- **Transfer address (i.e. Slot ID, QL#) to SDS.**
  - For Transfer or Search, nothing else happens during the T cycle.
  - For Adds, the Add Server reads data out of the Data Buffers to prepare to do a write.
    * This control can be done directly by the Add Server as no other Data Pipe contenders use these paths. Thus, these latches aren't strictly associated with a cycle point.
    * Note that 16 bytes are read out per cycle and concatenated to form 32 byte QLs. Because of this, the Add Server will only try to do a Write flow every other cycle.

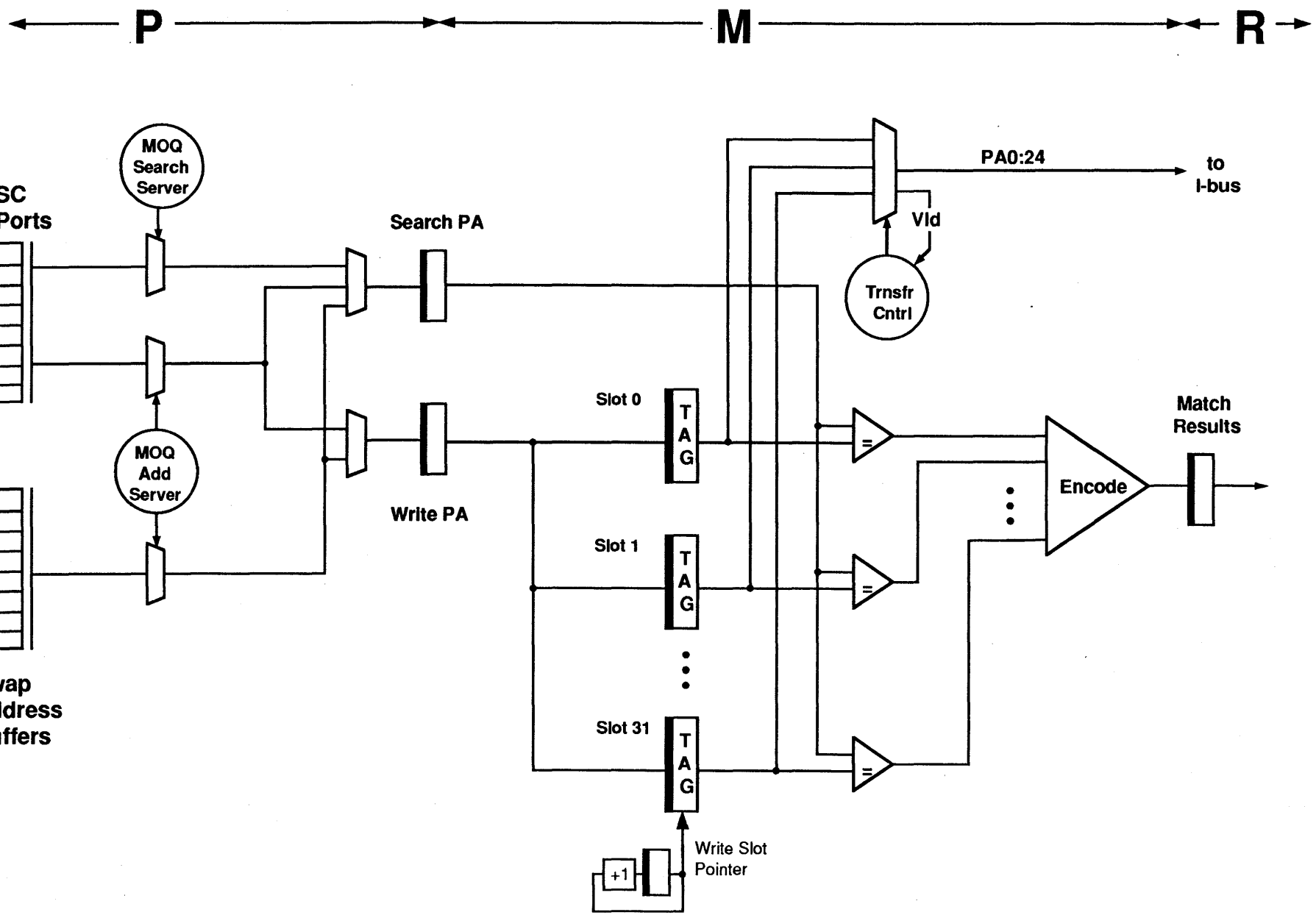
- **Distribute address within SDS.**

- **Access MOQ Strams.**
  - This cycle point is the address/data in latches in the STRAM macro.

- **Results available.**
  - The 32 bytes can be MUXed over to the MSU 16 bytes at a time, but each 16 bytes takes 2 cycles to transfer, so the Transfer Controller does reads every 4 cycles.
  - Note the MOQ Bypass path back to the Fetch Data Buffers.

MOQ TAGs

Rev. 1, 8/91



MOQ TAGs

- **TAGs implemented in latches, allowing some special capabilities.**
  - Can match against all 32 TAGs in parallel.
  - Can do concurrent Reads and Writes.
  - Transfer Controller has its own selector to read TAGs.
  - Valid bit Resets dedicated to Transfer Controller (not shown).

- **Transfer Controller**
  - Loads data into MSDIR, then _____
  - Once accepted into the I-bus, it resets the Valid bit.

- **Pipelined TAG Access for Add and Search Servers**
  - PMR pipe
  - Transfer Controller owns all the resources it needs, so it doesn't need pipe access.

Priority cycle

- Search and Add Servers contend for pipeline access.

Match cycle

Search PA register

- Matched against all 32 TAGs in one cycle.
- Used by Search Server to _____
- Also used by Add Server to _____
- Search results encoded and latched.

Write PA register

- Contains PA to be written into next MOQ slot.
- Owned by Add Server, so it's not strictly part of the pipe.

Result cycle

- Match Results available for examination.

Swap LMO MOQ Add Timing

```
LMO1              P A T B R
LMO2                      P A T B R
QWs in SDB                          0 1 2 3 4 5 6 7
_____________________________________________________
Write QL0                             P T D A R
Write QL1                                P T D A R
Write QL2                                   P T D A R
Write QL3                                      P T D A R

PA SwpAdrBfr                       |----------------->
MOQ TAG Update                                |P-M-R|
```

Search Timing (match case)

```
Fetch in SC       |------------------------------->
Search Flow          P M R
Match found             |-|
Read QL0                   P T D A R
Read QL1                      P T D A R
Read QL2                         P T D A R
Read QL3                            P T D A R
*QWs in FDBs                          0 1 2 3 4 5 6 7
```

Transfer Controller Flows

```
Read QL0          P T D A R
Read QL1                  P T D A R
Read QL2                         P T D A R
Read QL3                                P T D A R
*QWs in MSDI                      |-0-|-1-|-2-|-3-|-4-|-5-|-6-|-7-|
I-bus Req                                           |-|
```

* Latch points may be missing from Block Diagrams. Timing diagrams should be correct.

MOQ Algs

Add Server

- The MO Server posts a status bit indicating the Swap LMO1 and LMO2 flows have (or soon will have) loaded the Swap/Store Data Buffers.
- Kicked by this, the MOQ Add Server will initiate 4 MOQ Write flows to write the data into the MOQ.
- In parallel with the write flows, the Add Server writes the PA (from the Swap Address Buffer) into the corresponding MOQ TAG.
- The add server also checks the TAGs for an older copy of the line to invalidate.

Search Server

- Fetch requests need to check the MOQ to see if the data's there.
- One TAG search flow examines all 32 TAGs.
- In the case shown it happens to get a match.
- The Search Server requests priority for 4 data read flows and loads the data into the Fetch Data Buffers, for subsequent delivery to the requester by the MI server.

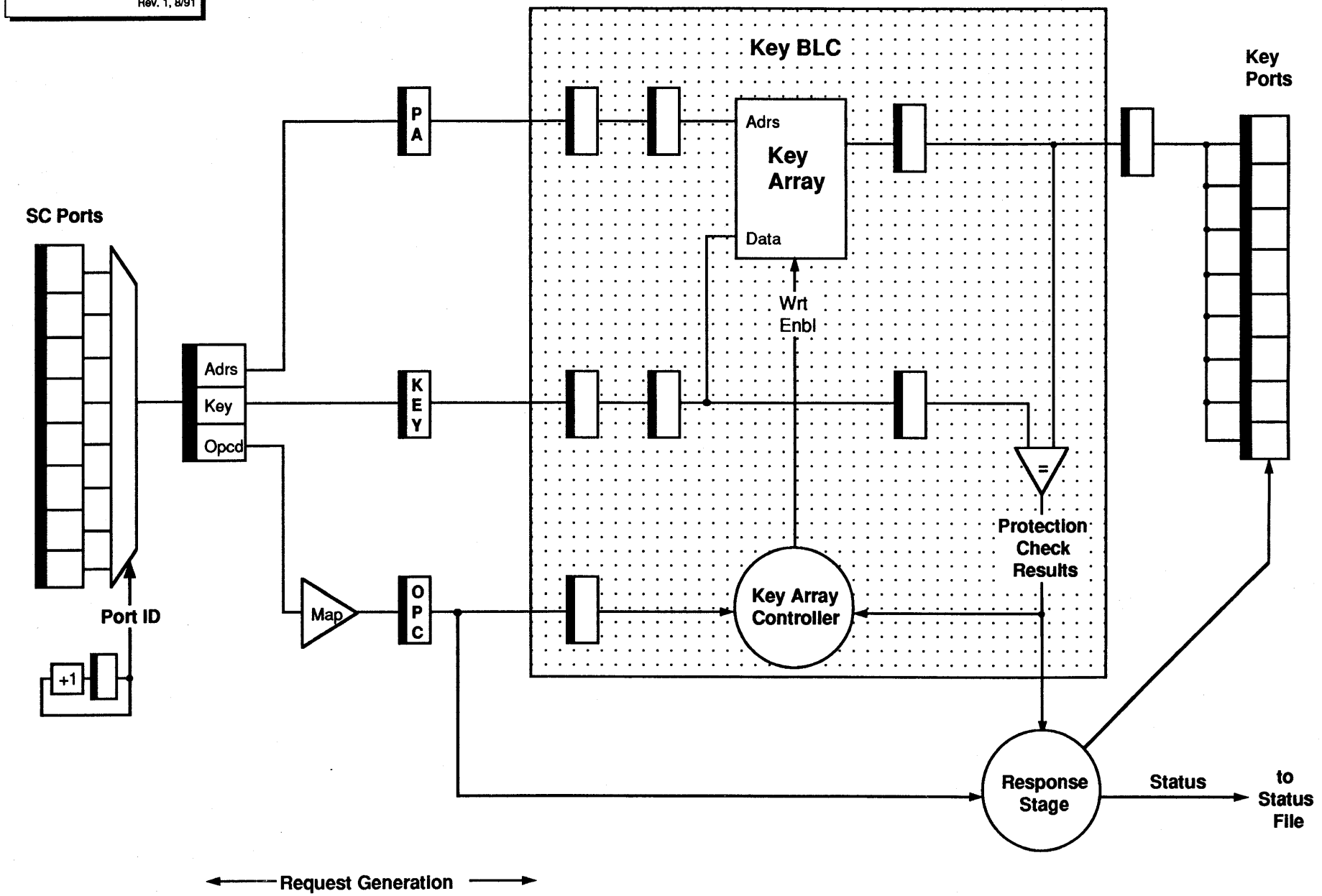
Note - the timing diagram takes into account latch points that aren't shown in the block diagrams.

Transfer Controller

- Having found a valid MOQ entry in the background, the Transfer Controller initiates 4 read flows to read out the entry.
- The data is muxed 16 bytes at a time to the MSU, and each 16 byte transfer takes 2 cycles, so the TC does read flows every 4 cycles.
- In the MSU, the data is loaded into 1 of 2 MS Data In Registers.
- Once all the data is read out, the TC requests I-bus priority to send a MS Write request to the MS Server.
  - * Note - this request is sent before all the data is actually in the MSDIR. The timing is such that, by the time the MS Server does the actual write, all the data will be there.

Key Server

Rev. 1, 8/91



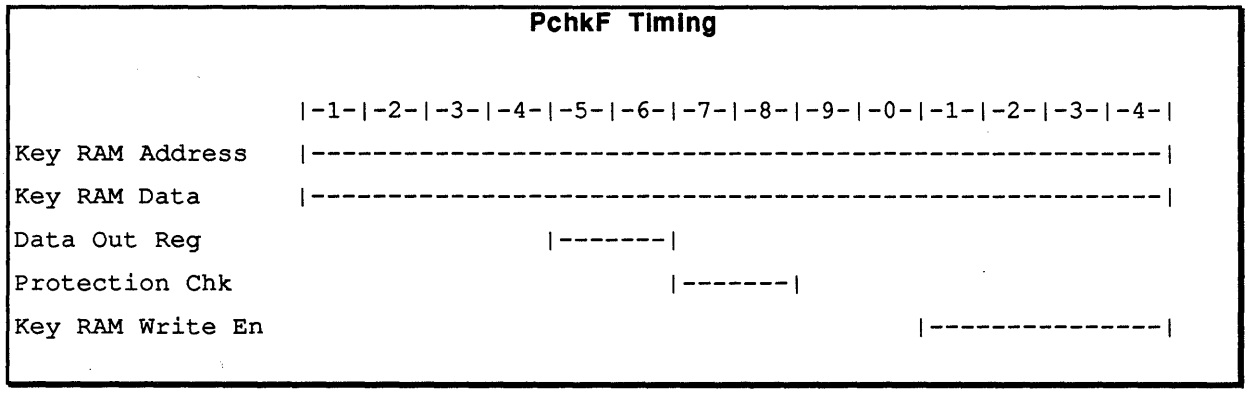
Key Server

Request Generation Stage

- Recodes SC opcode to an internal key opcode. Includes (among others):
  - * SRB — Set Reference Bit.
  - * SRCB — Set Reference and Change Bits.
  - * STORE — Write entire key.
  - * FETCH — Read and return key.
  - * PchkF — Do Fetch Protection Check (using provided key). If OK, Set Ref. Bit.
  - * RRB — Read and return key, reset the Reference Bit.
- Fires up Key Array Controller with new opcode (except on NOP).

Key Array Controller

- Controls chip selects and write enables
- For opcodes doing both a read and a write, write enable is delayed until read is done.



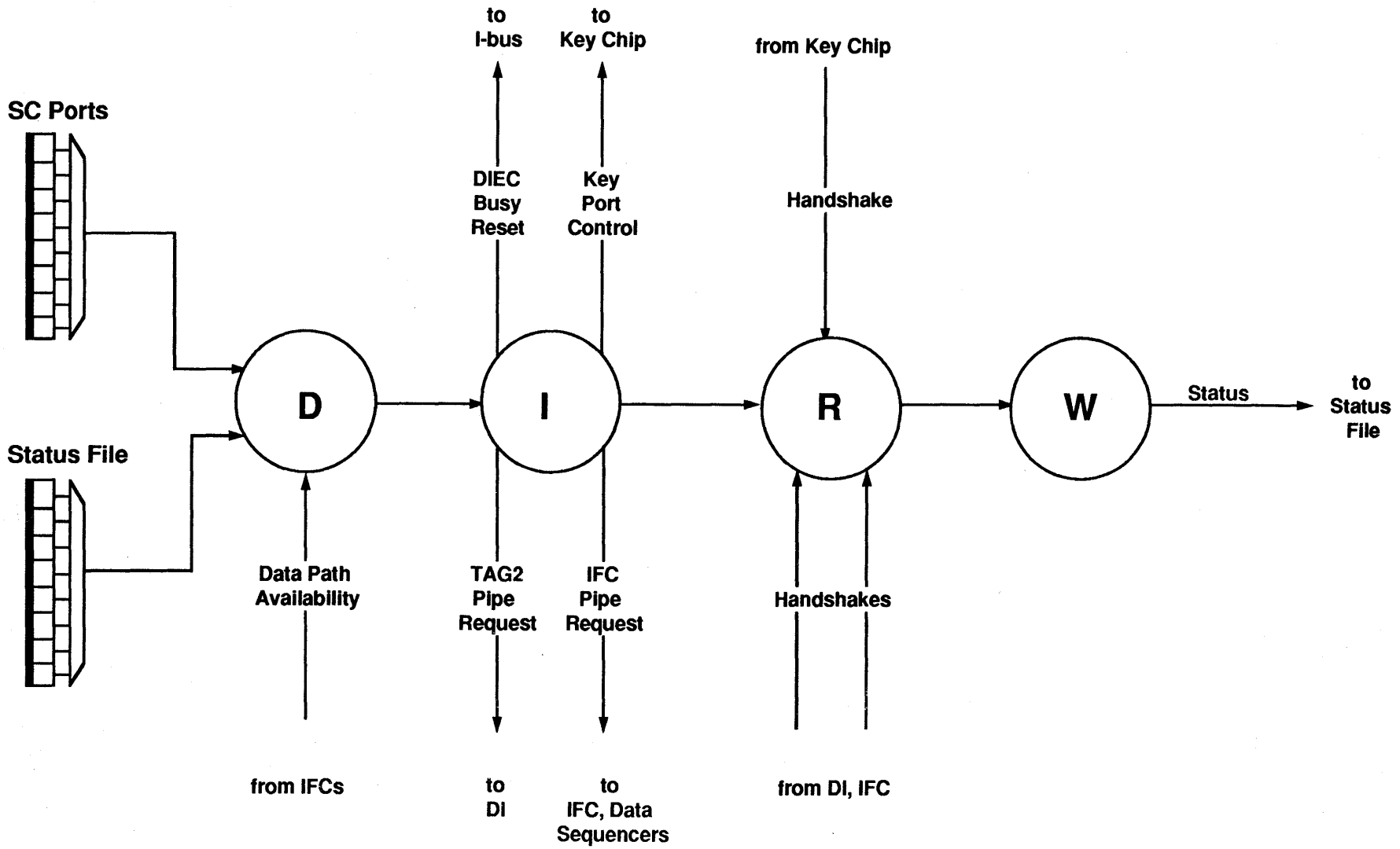
- Passes results on to Response Stage State Machine
- Array is _____ deep.
- Key Array runs on half-cycle clocks.

Response Stage

- On Key Reads, key is loaded into port.
- Sets status bits describing request results.

MI Server

Rev. 1, 8/91



MI Server

- **Pulls it all together and returns results to requestor.**
  - Because the MI server is the one that "wraps it all up", it's the prime reader of status bits.

- **D-cycle**
  - Waits for appropriate status bits to be set.
  - Based on status bits, initiates response by kicking off I-cycle.
    * For S-unit requests, maps the SC opcode to an S-unit opcode.

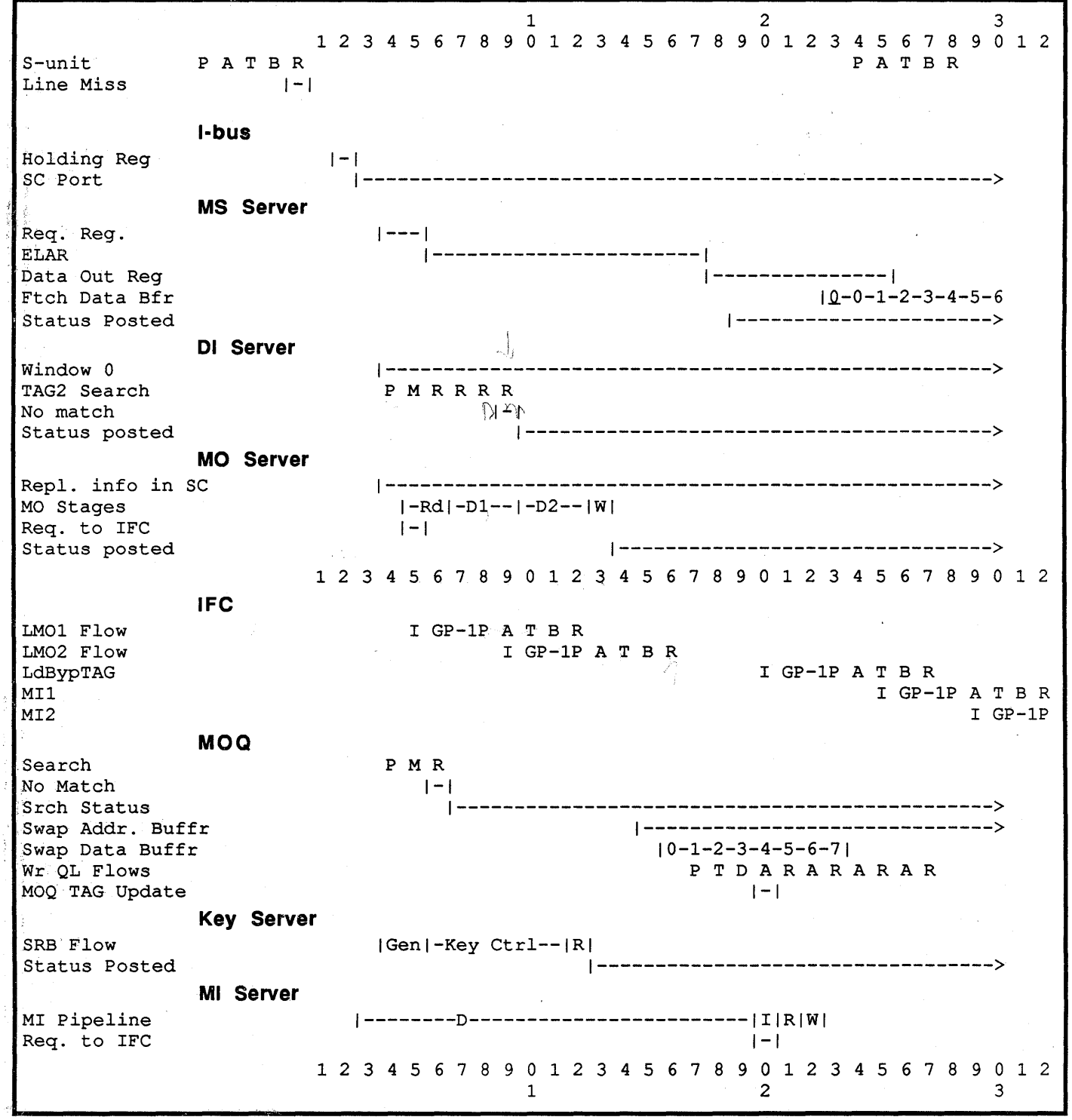
- **I-cycle**
  - Sends requests to various resources:
    * IFC to make S-unit requests.
    * Based on IFC grant, kicks off a Data Sequencer on the SDS to control data muxing.
    * DI to update TAG2.
    * Key chip to read key results out of the Key Ports.
    * I-bus to reset the DIEC busy.

- **R-cycle**
  - Receives handshaking from above resources to make sure there wasn't an error.
  - If there was, MI stops processing requests until S-code can fix things up.

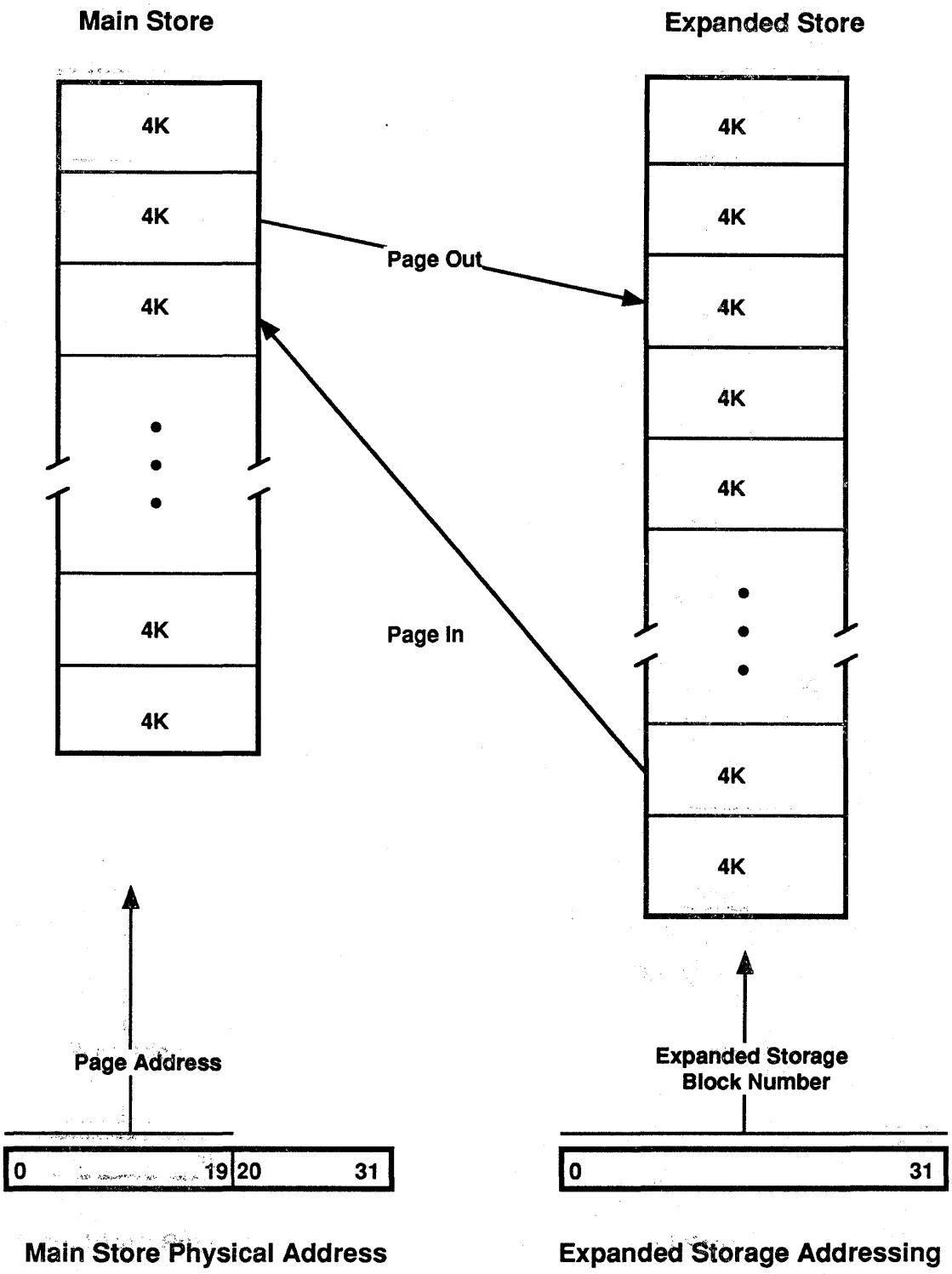
- **W-cycle**
  - Sets MI Done in the Status File.

Sample SU Fetch Request Timing

(semi-accurate!)



- This page intentionally left blank -



Expanded Storage Architecture

- **Large, slow, dense storage**
  - In SONA, 4 GB/side (4 Mb DRAMs). Later, 16 GB/side (16 Mb DRAMs)
  - Larger, slower than MS. Smaller, faster than disk.

- **Fundamental unit is a page (4K)**
  - In Sequoia a 32 bit Expanded Storage Block Number points to a page.
  - Allows up to 16 TB of data to be stored.
  - Since SONA maxes out at 32 GB, bits 0 96 are always zero.

- **Page Ops transfer a page between Expanded Store and Main Store**

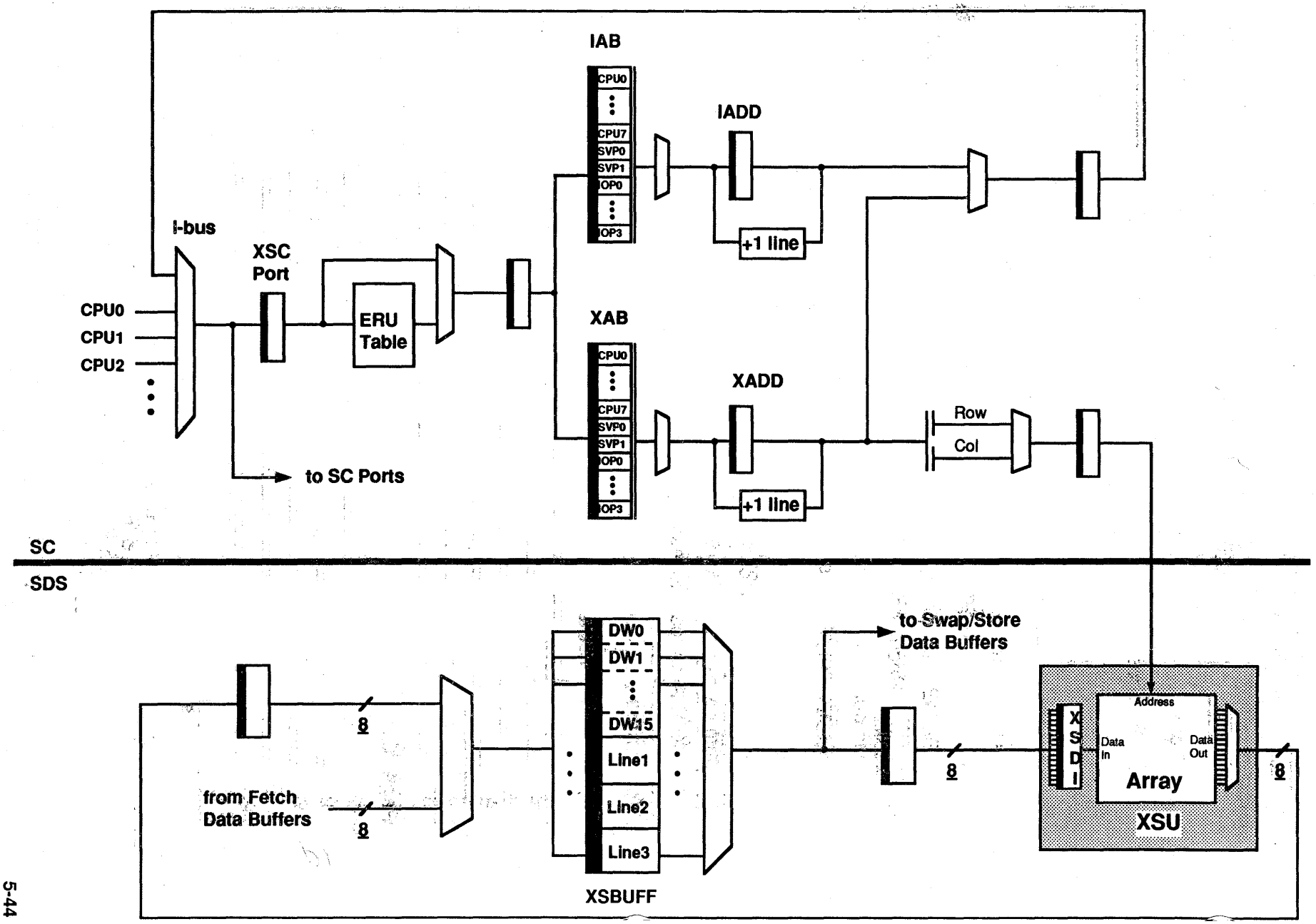
  Page In: Copies a page of data from XSU to MSU.
  Page Out: Copies a page of data from MSU to XSU.

  - Instructions include a Main Store address and an ESBN.
  - Operation is synchronous; the CPU waits until the transfer is complete.

- **Naming confusion**
  - IBM calls it Expanded Store.
  - Many people call it Extended Store.
  - Both ESU and XSU are used as acronyms.

XS Address and Data Paths

Rev. 1; 8/91



XS Address and Data Paths

- **XSC Port**
  - I-bus sends XS requests to dedicated XS Controller Port.
  - A Page Op requires 2 I-bus commands, one for each address:
    * The ESBN goes through the ERU Table, then into the XAB.
    * The Main Store PA bypasses the ERUT and goes into the IAB.
- **Address Buffers**
  - Two sets of buffers, one for I-bus Addresses (IAB) and one for XSU Addresses (XAB).
  - IAB/XAB each provide 1 dedicated buffer for each possible requestor (CPU, SVP, IOP).
    * Each Requestor will only have 1 Page Op pending at a time.
  - One operation is handled at a time. Any others wait in the ABs until processed.
  - ABs are ***not*** a queue. The XS Controller processes them round robin.
- **Data Buffers**
  - Buffering provided for 4 lines of data.
  - Data paths are 1 DW (8 bytes) wide.
    * XSU Array path takes 2 cycles to transfer 1 DW.
- **Algorithms**

Page In

1. _____
2. _____
3. _____
4. _____
5. _____
6. _____

Page Out

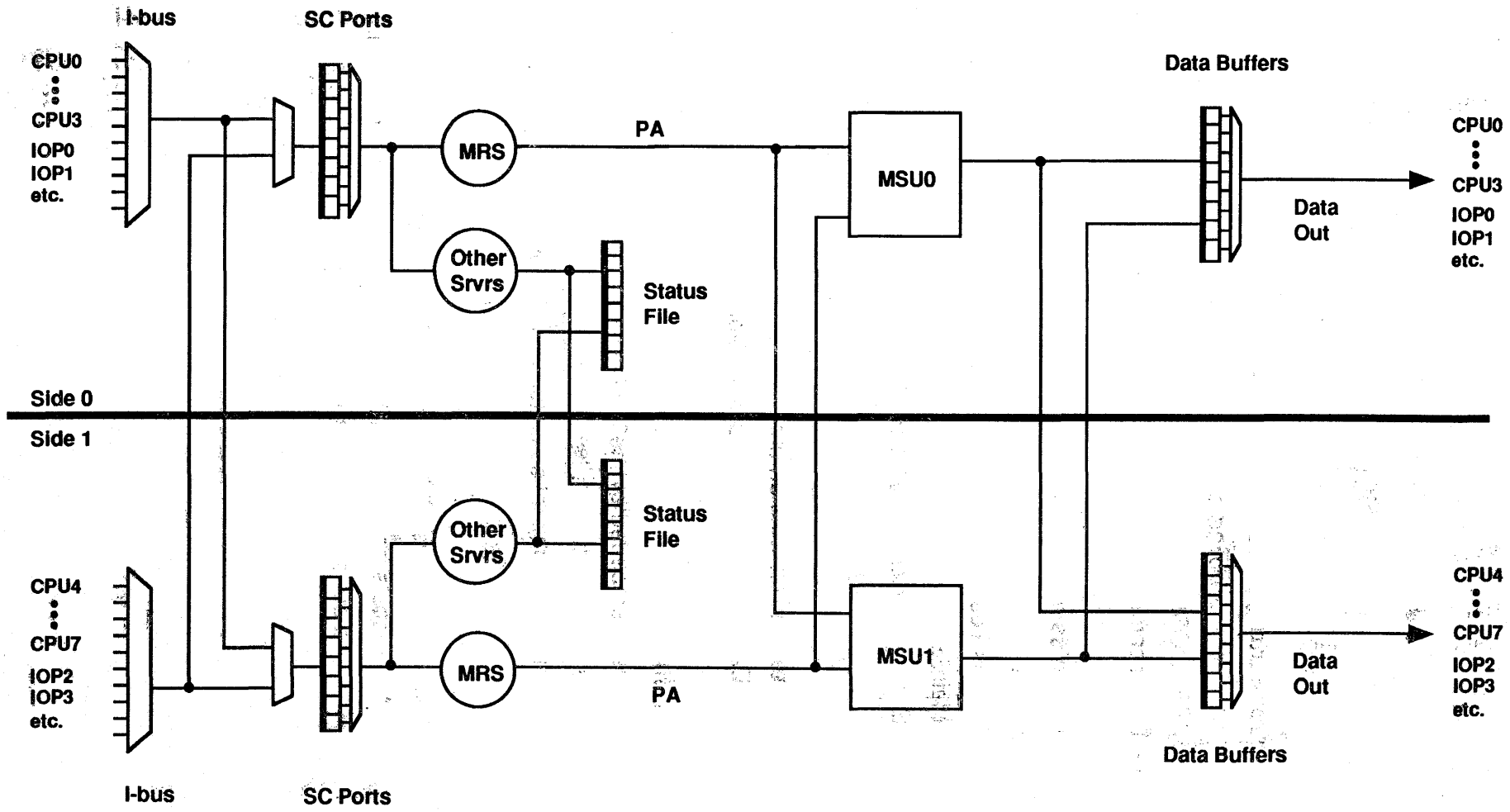
1. _____
2. _____
3. _____
4. _____
5. _____

Other Algs

- SVP can do line fetches/stores.
- IOP Page Ops provided in anticipation of asynchronous page in/out.
- MSU-MSU copies also implemented to support dynamic reconfiguration.
- Background refresh done for DRAMS.

System Storage DS Concept

Rev. 1, 8/91



System Storage DS Concept

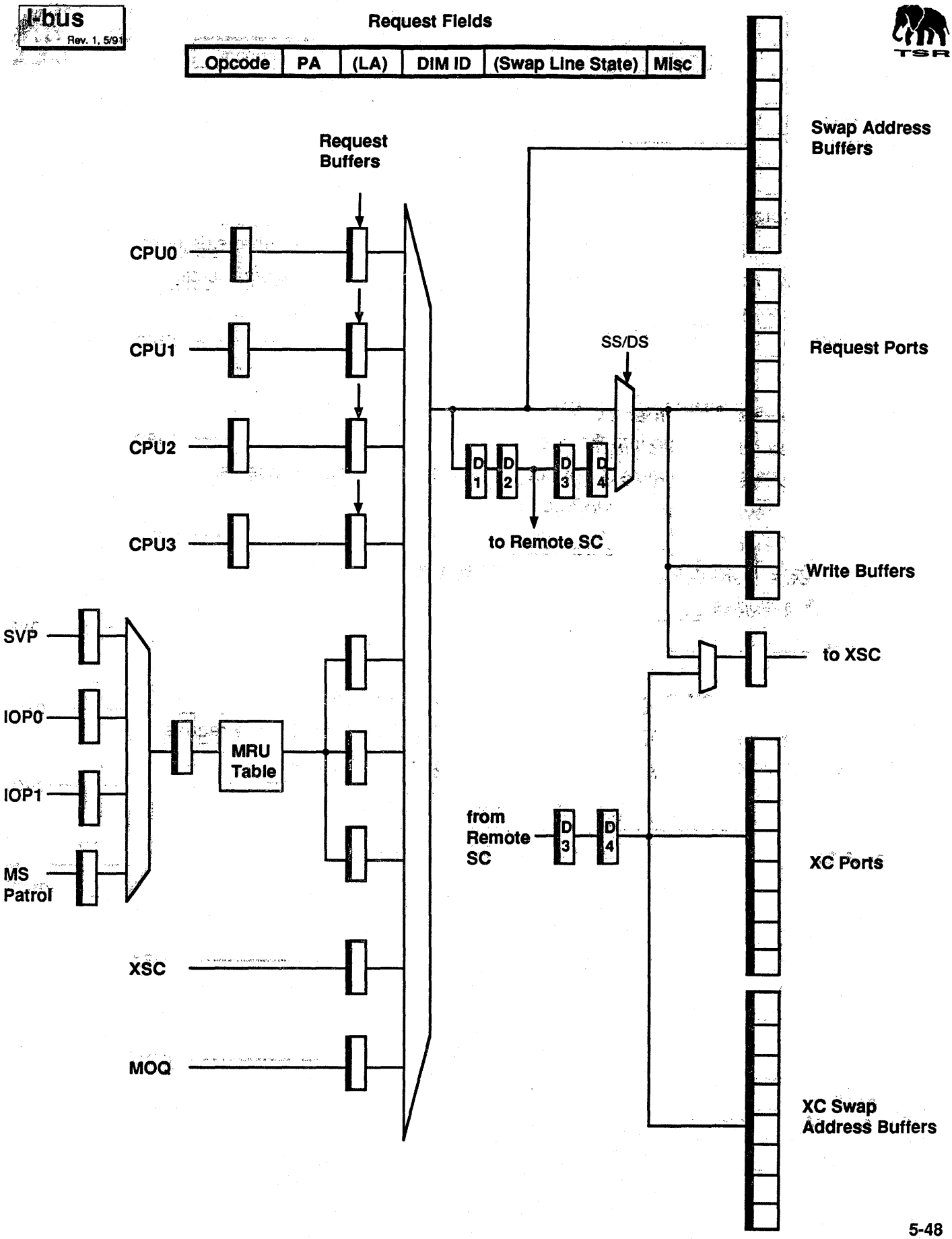
- **System Storage is DS Focal Point**
  - All cross-coupling done here.
  - CPUs, IOPs only talk to their local SC/SDS.
  - Data can be on either side of a DS system.

- **Approach is to cross-couple at key points within System Storage**
  - I-bus
  - Status File
  - Main Store
  - Some other cross-coupling isn't shown.

- **I-bus**
  - Same request wins on both sides.
  - DIEC and other busies must be kept in synch.

- **Status files**
  - Status bits needed by the other side (e.g. DI search results) are cross-coupled.
  - Done by the status file itself.
  - Means duplicating these status bits to provide storage for status from local and remote.

- **MSU**
  - Dual ported, either SC can access both MSUs.
    - MSU inputs have a selector to pick which side drives them.
  - Two data out paths, one for each SDS.



I-bus DS Design

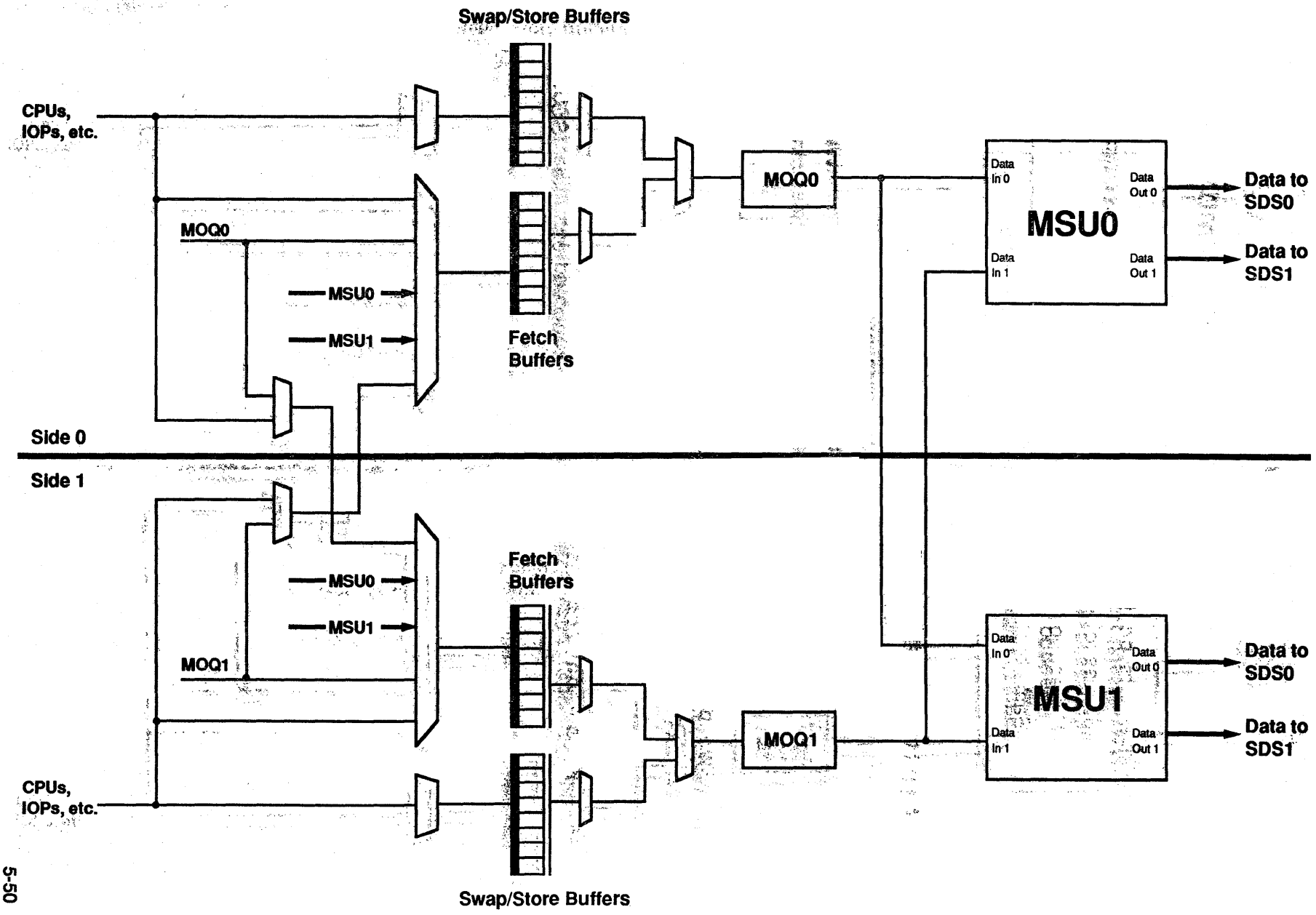
- **Four cycles added in DS mode to allow for cross-coupling**
  - First two latch points "early-up" the request to send to the other side.
  - Next two latch points "normalize" the late request from the other side.
  - Winning request(s) loaded simultaneously on either side.

- **Remote requests loaded into Cross Couple Ports**
  - Local requests go into ReQuest Ports.
  - Allows some servers to not bother even seeing remote requests.
    * They just look at the RQ ports.
    * Includes _____.
  - Other servers need to alternate between the two sets of ports.
    * Includes _____.

- **Element and DIEC busies are same on each side**
  - Both I-buses set the busies at the same time when loading the request.
  - I-bus cross-couples the resets so they go off at the same time.

Data Cross Coupling in DS

Rev. 1, 8/91



Data Cross Coupling in DS

- **Move-outs go to MOQ on same side as the requesting CPU.**
  - Bypass to Remote provided for DI MOs.
    - * Needed if _____.
  - Same bypass path used for MOQ bypass to remote.

- **MSU data-in dual ported.**
  - MOQ loads Data In Register of appropriate MSU.
  - Then sends MS Write request through I-bus, into its local Write Buffer.
  - The local MRS will send the address and write enable to the target MSU.

- **Two data out paths provided.**
  - One for each SDS.

- **Keys (both address and data) cross-coupled just like MSU.**

Things to Remember from the Class

1.

2.

3.

4.

5.