18-544: Network Design and Evaluation

IXP2400 Architecture and Programming

August 31, 2005

Agenda

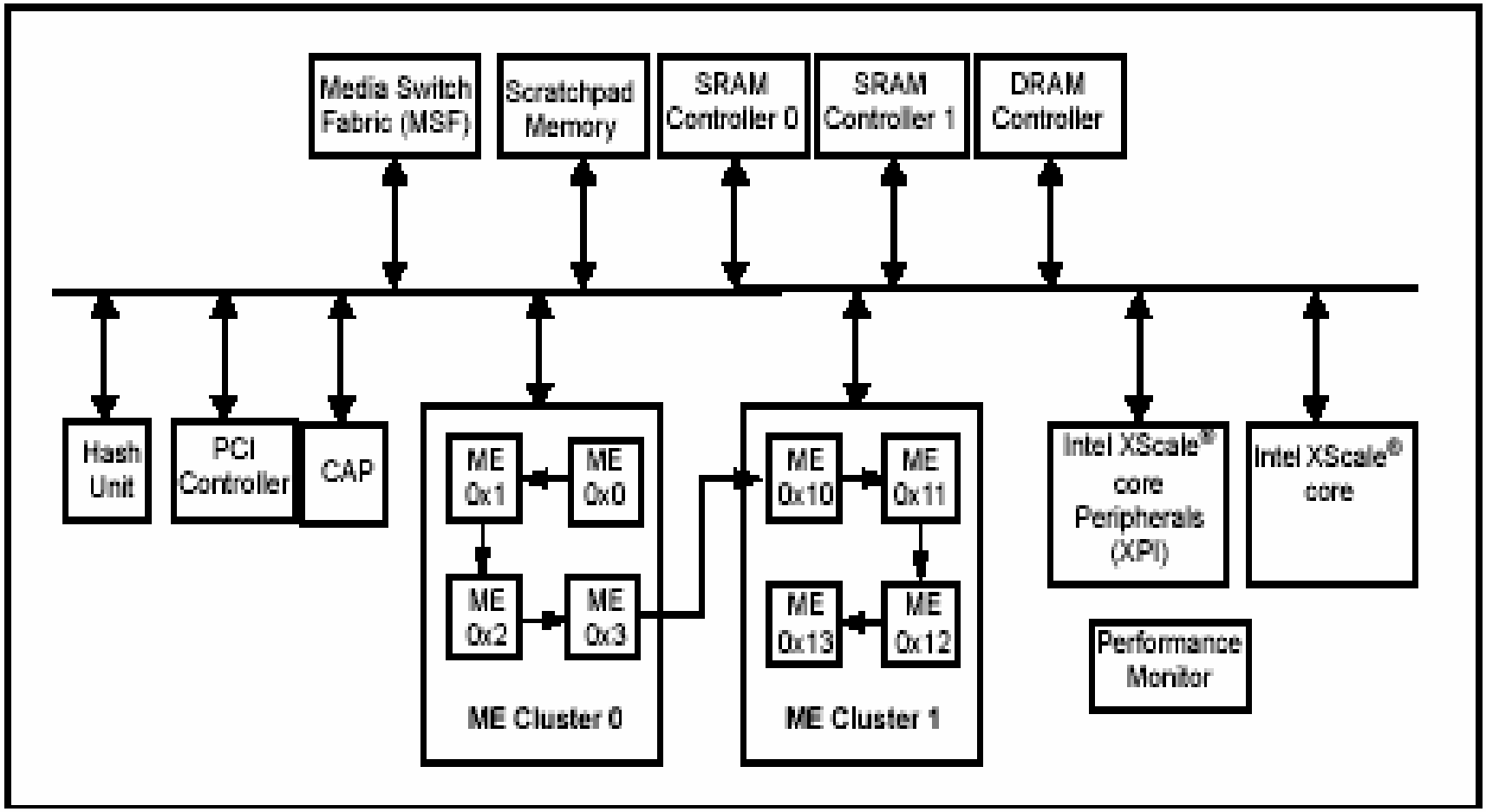
- Reminder: optional lab session this Friday
  - Hamerschlag Hall 1204, 2:30 pm
  - Introduction to IXP programming
    - Development environment
    - Initial code base
- Recap of last lecture
- IXP2400 architecture overview
- MEv2 instruction set crash course
- Data plane programming
- Distribute the IXA SDK cds

Recap

- Network processors
  - Optimized for network workloads
  - High-speed data movement
  - Support for common network functions
- Architectural Trends
  - Parallelism
    - Multi-core
    - Multi-threaded
  - Intelligent bus and memory interfaces
    - Asynchronous operation

IXP2400 Architecture Overview

IXP2400 Block Diagram



Media and Switch Fabric Interface (MSF)

- Primary data path interface
  - Connects IXP2400 to external devices, e.g., MAC
  - Independent Rx and Tx interfaces
  - Separately configurable for various bus protocols (e.g., UTOPIA, POS-PHY)
- Connected to internal busses
  - Microengines
  - DRAM controller
- Data RAM's
  - 8K RBUF and TBUF
  - Packets segmented into fixed-size buffers

DRAM

- Off-chip packet data memory
  - High latency
  - High bandwidth
- DRAM Controller
  - “read” and “write” commands from requestors
  - Burst transfers
    - 16-128 bytes
    - 64-bit transactions

SRAM

- Off-chip application data memory
  - E.g., route table
  - Lower latency than DRAM
  - Lower peak bandwidth
- 2 independent SRAM controllers
  - Maintain command queues from requestors
  - 32-bit transactions

SRAM commands

- Reads and Writes
  - 4-64 byte transfers
- Atomic operations
  - Test-and-set, increment/decrement, swap, etc
  - Atomic with respect to other atomic operations only
- Queue operations
  - SRAM-based queue-like data structures
    - Free buffer list
    - Multi-buffer packets
  - Enqueue and dequeue commands
  - Linked list implementation
- Ring and Journaling Operations
  - Fixed-size circular buffers
  - Insert and remove commands

Scratchpad

- 16K on-chip shared memory
- Low latency
- Frequently-accessed application data
  - E.g., packet counters
- 32-bit transactions

Scratchpad Commands

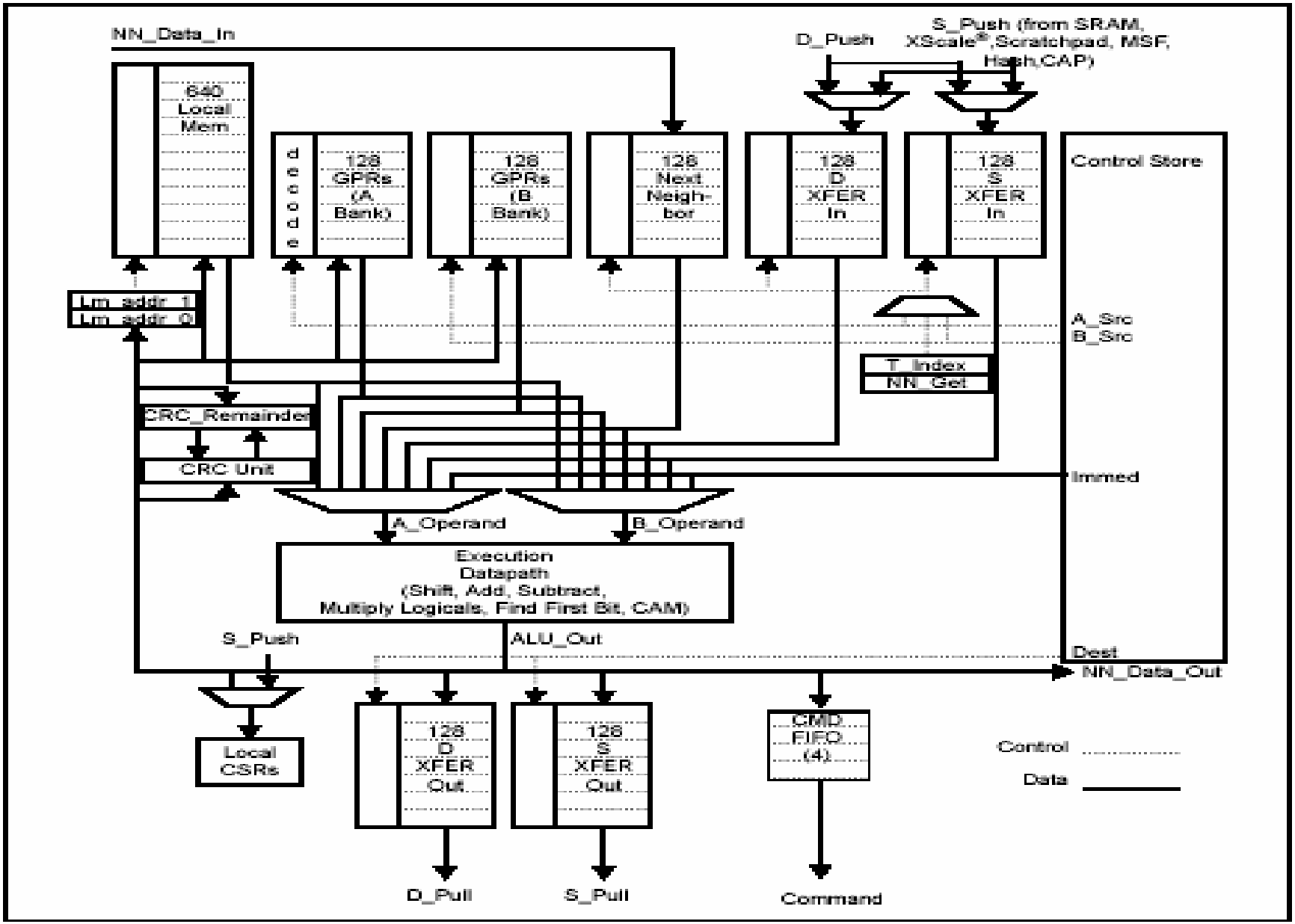
- Reads and Writes
  - 4-64 byte transfers to/from Microengines
- Atomic Operations
  - Increment/decrement, add/subtract, swap, etc
  - Atomic with respect to normal reads/writes as well
- Ring Operations
  - Up to 16 rings
  - 128, 256, 512 or 1024 32-bit words
  - Naturally aligned in Scratchpad address space
  - “Get” and “put” commands

XScale Core

- Standard embedded CPU
- Runs Linux and control plane software
- More on this next time

Microengine Architecture and Programming

Microengine Block Diagram



Register Files

- General-Purpose Registers
  - 256 per microengine
  - Split in two banks
- Transfer Registers
  - Access to all external resources
  - DRAM transfer registers: 128 in, 128 out
  - SRAM transfer registers
    - 128 in, 128 out
    - All resources other than DRAM (SRAM, Scratchpad, MSF)
- Next-Neighbor Registers
  - 128 per microengine
  - Another form of IPC
  - Neighboring microengines have dedicated connections
  - Can be used for local storage

Local Memory

- Private microengine memory
- 640 32-bit words
- Lowest latency in memory hierarchy
- Separate address and data accesses
  - 3-cycle latency before address becomes effective

Control Store

- Static instruction memory
  - 4096 40-bit instruction words
  - No i-cache
  - Potential code size concerns
- Loaded by XScale at initialization time
  - XScale must shutdown and restart ME to reload control store

Contexts and Addressing

- 8 hardware threads per microengine
  - Each thread has own PC
  - Fast context switching
- Register Usage
  - Context-relative addressing
    - “private” registers
    - 32 GPR’s per thread
    - 32 SRAM, 32 DRAM transfer registers per thread
    - 16 Next Neighbor registers per thread
  - Absolute addressing
    - GPR’s only
    - Threads access same physical register

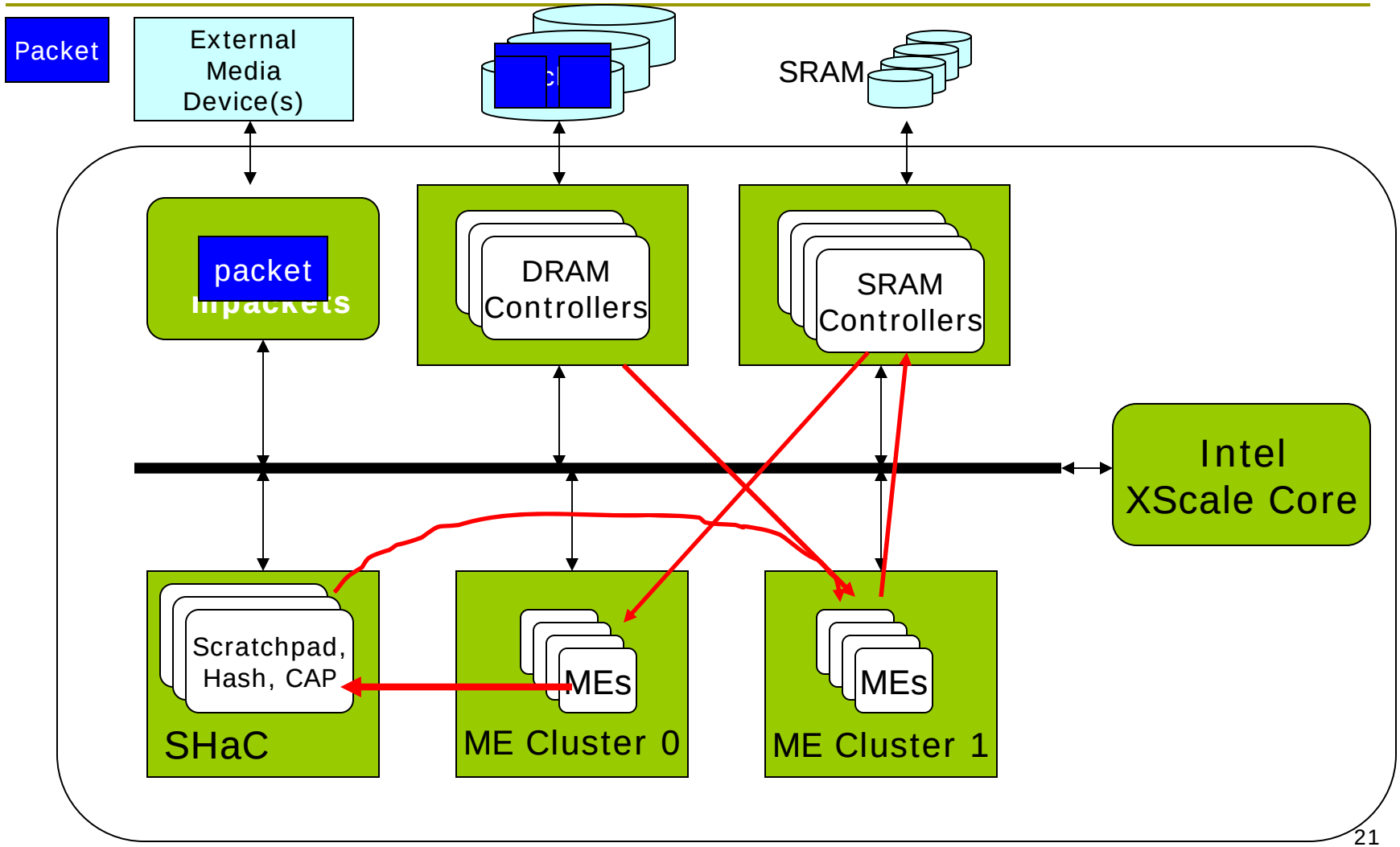
Synchronization and Signals

- Asynchronously-completing external events
  - E.g., memory references
  - ME's will not stall on in-flight register writes
- Signals
  - Used to indicate event completion to ME
  - 15 signals per context
  - I/O instructions specify signal to use
    - Thread swapped out and woken up upon completion
    - Threads can explicitly poll a particular signal

Other Hardware

- CRC Unit
  - Efficient CRC computation
- CAM
  - 16 32-bit entries
  - Can be used to implement small cache

Example


Packet
External Media Device(s)
SRAM
packet
DRAM Controllers
SRAM Controllers
Intel XScale Core
Scratchpad, Hash, CAP
SHaC
MEs
ME Cluster 0
MEs
ME Cluster 1

MEv2 Instruction Set

Assembler Syntax

- Register declarations

```
.reg $val1, $val2    ;SRAM transfer registers
.reg $$dramreg       //DRAM transfer register
.reg count, tmp      /* relative mode GPR's */
.reg @myreg          ;absolute mode GPR
.reg n$neighbor      //next neighbor register
```

- Transfer register ordering

```
.xfer_order $val1, $val2   // $val1, $val2 contiguous
```

- Signal declarations

```
.sig sig_scratch_write
```

ALU Instructions

```
alu[dest, A_op, alu_op, B_op]
alu_shf[dest, A_op, alu_op, B_op_shf]
```

- A_op, B_op
  - Input arguments: GPR, transfer, immediate
- Dest
  - Destination: GPR, transfer, neighbor
- alu_op
  - +, -, +carry, B-A, B, ~B, AND, XOR, etc
- B_op_shf
  - Shift applied to B argument
  - Left/right shifts, rotates
  - Specified as immediate or indirect

ALU Example

```
.reg tmp, ip_max_main
.reg in_d_netmask, in_d_network_addr
…
…
…
;get last ip adress of subnet & restore MSB of subnet in case
;it's clear to calculate using a correct netmask

alu[ tmp, --, ~b, in_d_netmask ]          ;tmp=~netmask
alu[ tmp, tmp, AND~, 1, <<31 ]            ;tmp&=~(1<<31)

;ip_max_main=tmp|network_address
alu[ ip_max_main, tmp, OR , in_d_network_addr ]
```

Branch Instructions

- Conditional branches

  b*cc*[label#], opt_tok

- Unconditional branches

  br[label#], opt_tok

- Other branches

```
br_bclr[reg, bit_p, label#], opt_tok
br=ctx[ctx, label#], opt_tok
br=byte[reg, byte, b_val, label#], opt_tok
br=signal[sig_name, label#], opt_tok
br_inp_state[state, label#], opt_tok
```

Branch Parameters

- Condition codes
  - `beq, bne, bge, blt, bge, bcc, etc`
- Optional token
  - Delay slots
  - `defer[n], n=1, 2, 3`
  - Max number of delay slots determined by instruction type

Jumps

- Computed branches (e.g., jump table)

  `jump[src, label#], opt_tok`

- Parameters
  - `src`
    - Jump table offset
    - GPR
  - `label#`
    - Jump table base address
  - `opt_tok`
    - `defer[n]`
    - `targets[label1, label2, …, label`*n*`]`
      - Possible jump table locations

Control Flow Examples

- Example 1: loops

```
.reg foo
…
/* foo initialized to 0*/
…
loop#:
    alu[ foo, foo, +, 1 ]
    alu[ --, foo, -, LOOP_COUNT_CONSTANT]
    br!=0[loop#]
```

- Example 2: contexts

```
br!=ctx[0, death#]
…
/* this code executed by thread 0 only */
…
death#:
    ctx_arb[kill]            ;this instruction kills the thread that executes it
    nop
    nop
    nop
    nop
```

DRAM Access Instructions

```
dram[cmd, xfer, op1, op2, cnt], opt_tok
dram[cmd, --, op1, op2, cnt], tok, opt_tok
```

- Parameters
  - `cmd`: command code
    - `read`, `write`
    - `rbuf_rd`, `tbuf_wr` in second form
  - `xfer`: source/destination DRAM transfer register
  - `op1`, `op2`: restricted address operands
  - `cnt`: transfer size (1-8) in quadwords
  - `tok`: `indirect_ref` specifies RBUF/TBUF address
  - `opt_tok`: `sig_done[sig_name]`, `ctx_swap[sig_name]`

SRAM Access Instructions

```
sram[cmd, xfer, op1, op2, cnt], opt_tok
sram[cmd, xfer, op1, op2], opt_tok
sram[cmd, --, op1, op2], opt_tok
```

- Parameters
  - `cmd`: command code
    - `read, write`
    - `incr, decr, swap, add, etc`
    - `enqueue, enqueue_tail, dequeue`
  - `xfer`: source/destination SRAM transfer register
  - `op1, op2`: restricted address operands
  - `cnt`: transfer size (1-8) in longwords
  - `opt_tok`: `sig_done[sig_name], ctx_swap[sig_name]`

Scratchpad Instructions

```
scratch[cmd, xfer, op1, op2, cnt], opt_tok
scratch[cmd, xfer, op1, op2], opt_tok
```

- Parameters
  - `cmd: command code`
    - `read, write`
    - `incr, decr, swap, add, etc`
    - `put, get`
  - `xfer: source/destination SRAM transfer register`
  - `op1, op2: restricted address operands`
  - `cnt: transfer size (1-8) in longwords`
  - `opt_tok: sig_done[sig_name], ctx_swap[sig_name]`

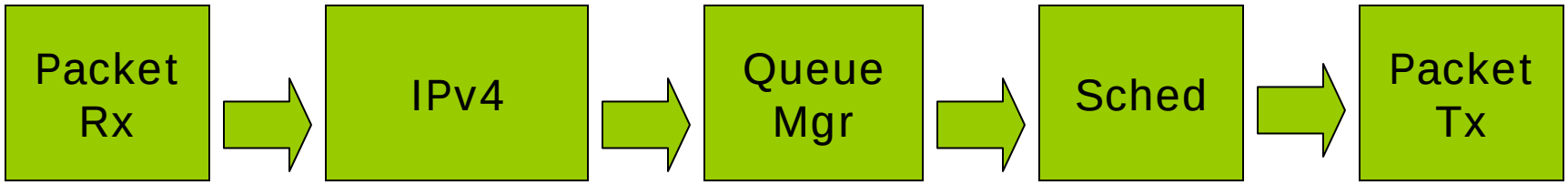
I/O Examples

```
// sram reads 4-byte words

.begin
     // register declaration
     .reg sram_address, $sxfer1, $sxfer2
     .xfer_order $sxfer1, $sxfer2

     immed[sram_address, 0x20]     ; init sram_address
     immed[$sxfer1,0xA0]           ; init $sxfer1 reg

     sram[write, $sxfer1, sram_address, 0,1],
ctx_swap[sram_sig_1]

     sram[read, $sxfer2, sram_address, 0, 1],
ctx_swap[sram_sig_1]

.end
```

Data Plane Programming Models

Microblocks

- Data plane software components
- Perform isolated network function
  - IPv4 forwarding
  - Packet Rx
  - Organized in “pipeline”
  - Communicate with adjacent microblocks and control plane

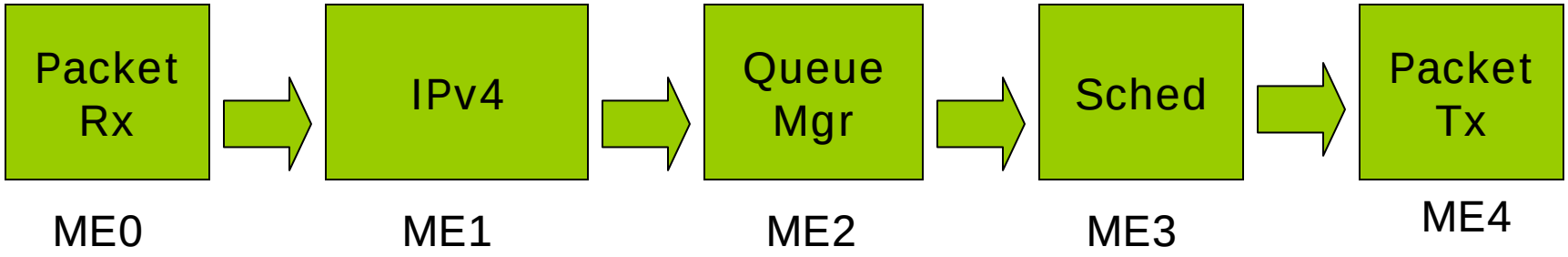


Microblock Groups

- Multiple microblocks on same microengine
- Packet transit logical, no inter-ME communication
- Organized in Dispatch Loop
  - Infinite loop
  - Microblocks executed in order for each packet

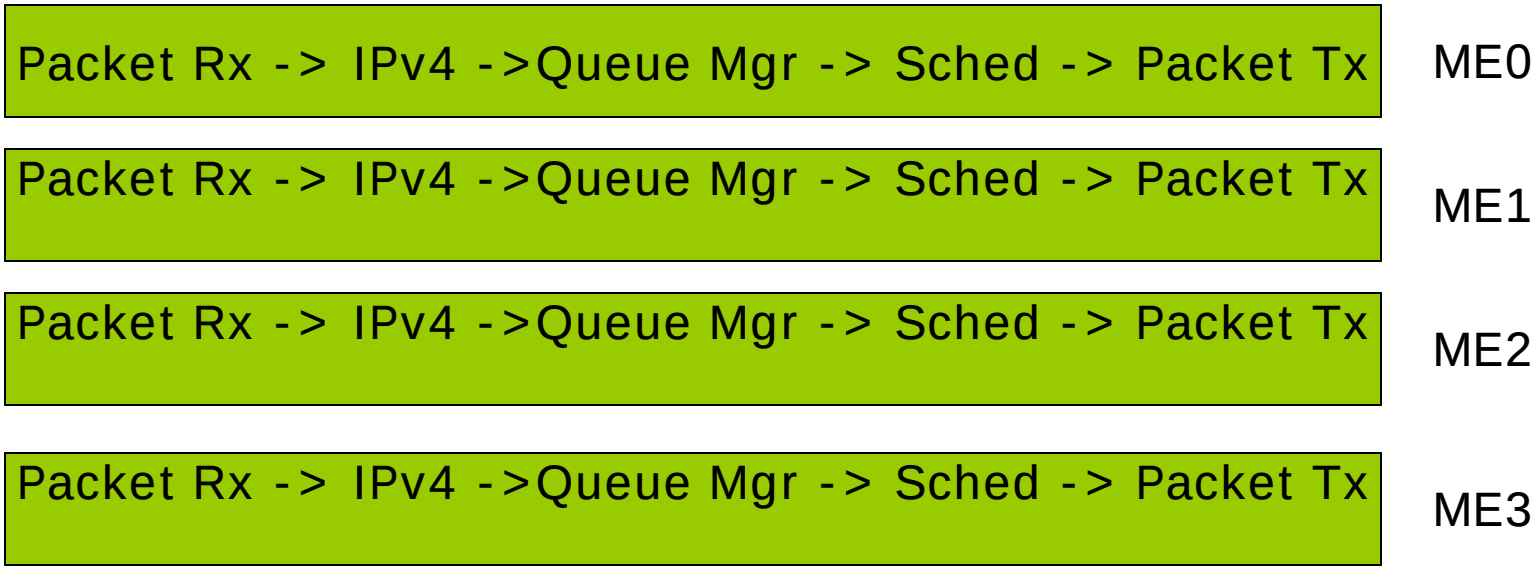
Pipeline model

- Packet processing spread across multiple microengines
- AKA “Context Pipeline”
- Extreme case: one Microblock per ME
- Packet physically moves through pipeline

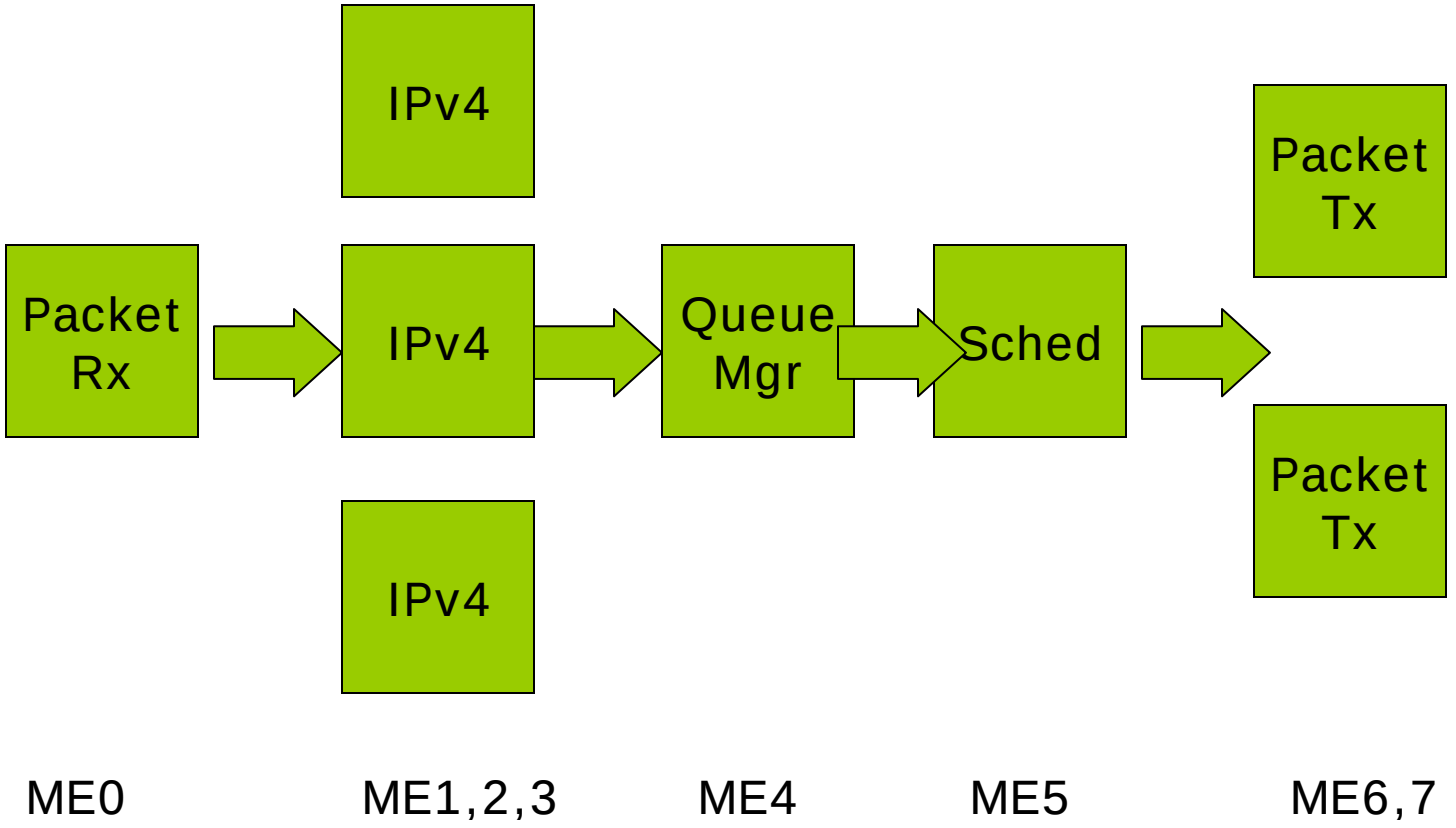


Pool-of-Threads model

- Each thread fully handles one packet
- AKA “Functional Pipeline”



Mixed Models: Pipeline of Pools



References

- IXP 2400/2800 Programming: Chapter 1 and 2
- Programmer's reference manual
- Hardware's reference manual