ECE 697J – Advanced Topics in Computer Networks

Embedded Control Processor

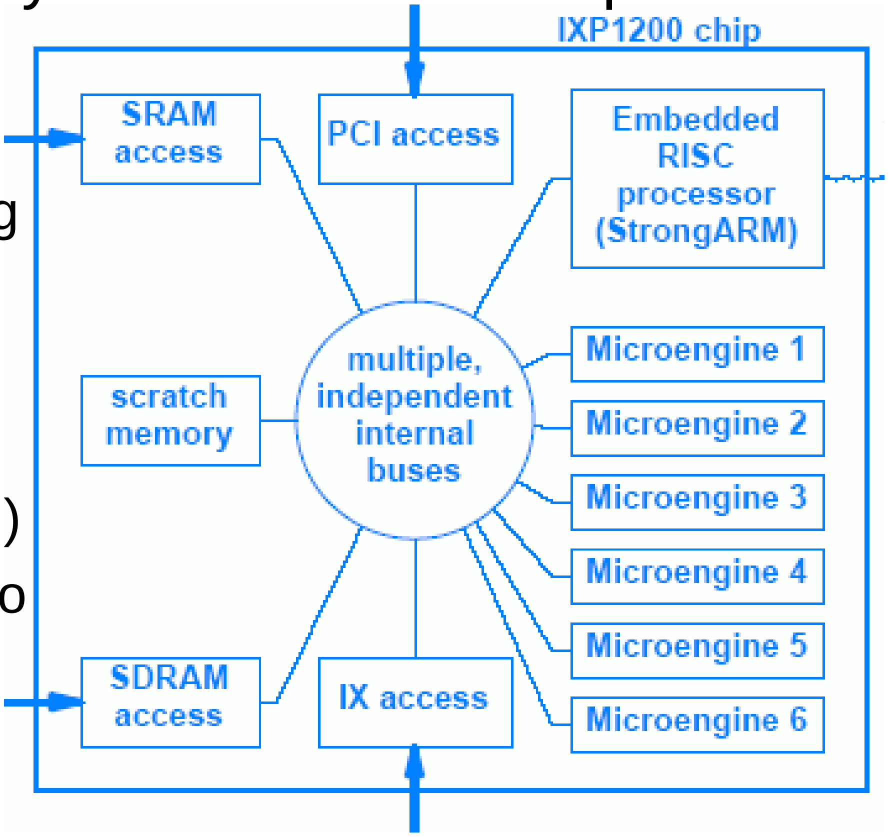
11/04/03

Overview

- More details on control processor (StrongARM)
  - Overall architecture
  - Typical functions
  - Processor features
- Microengines
  - Architecture and features
  - Differences to conventional processors
  - Pipelining and multi-threading

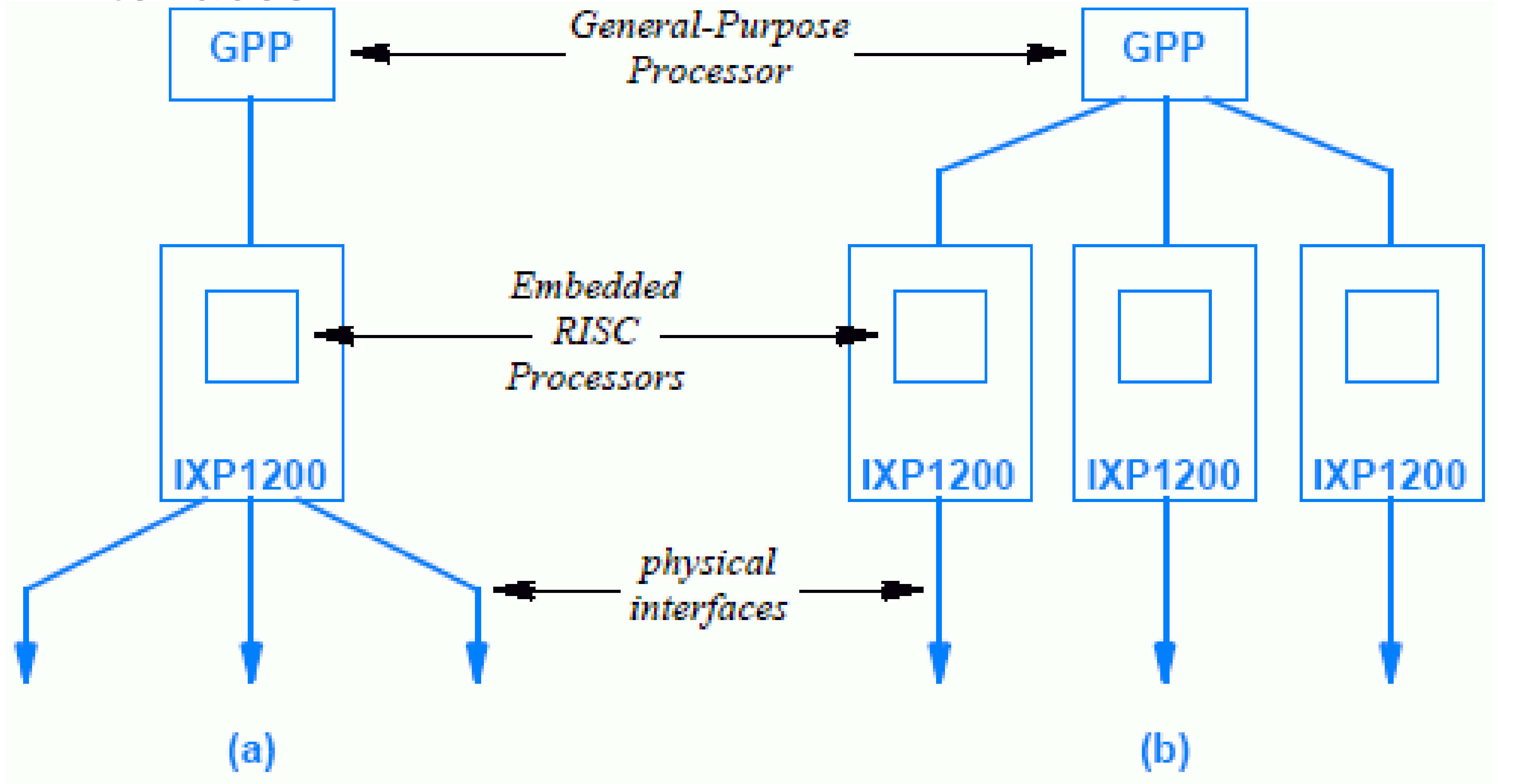
Purpose of Control Processor

- Functions typically executed by embedded control proc:
  - Bootstrapping
  - Exception handling
  - Higher-layer protocol processing
  - Interactive debugging
  - Diagnostics and logging
  - Memory allocation
  - Application programs (if needed)
  - User interface and/or interface to the GPP
  - Control of packet processors
  - Other administrative functions



System-level View

- Embedded processor can control one or multiple interfaces:



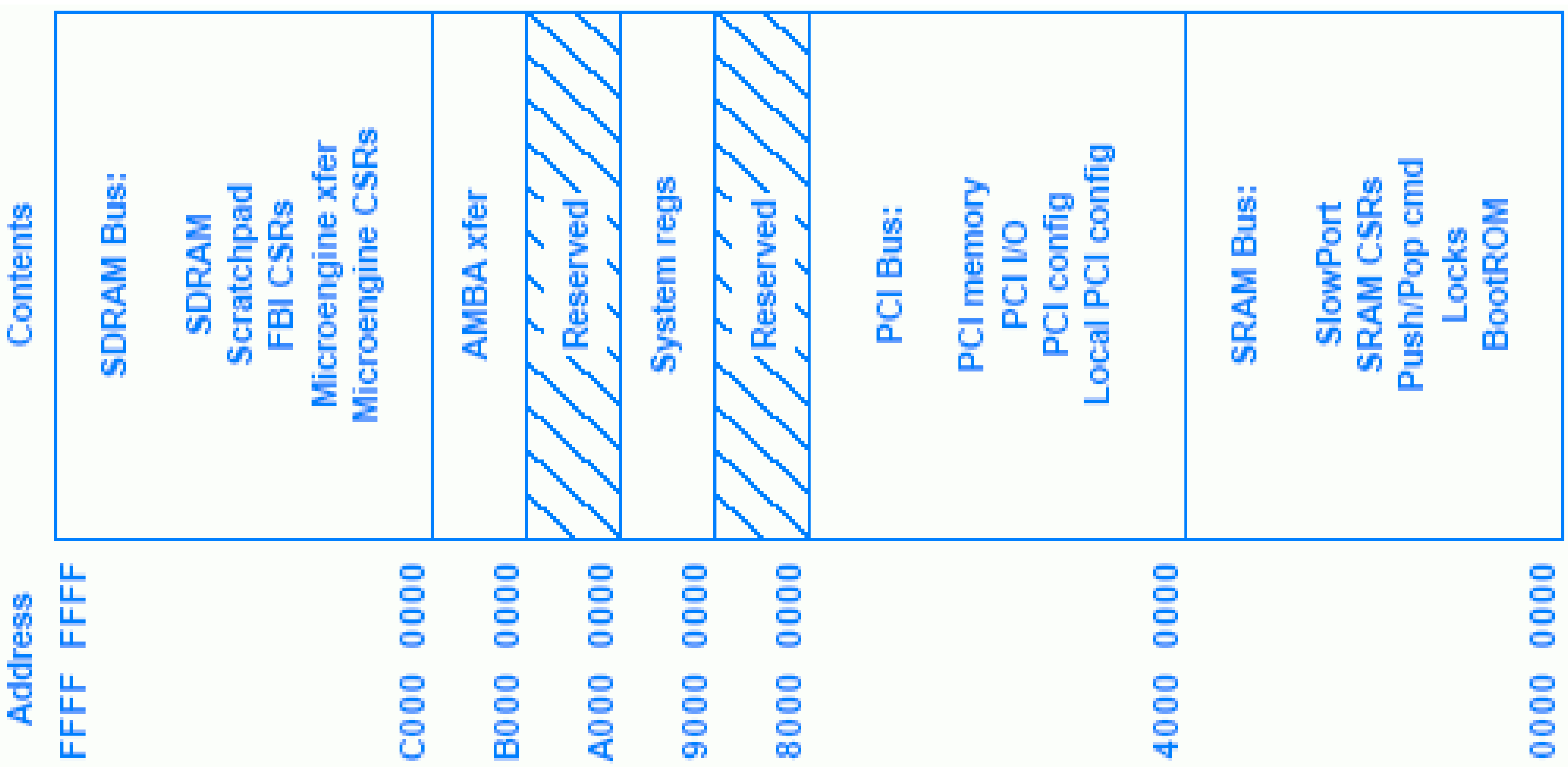
StrongARM Architecture

- ARM V4 architecture with:
  - Reduced Instruction Set Computer (RISC)
  - Thirty-two bit arithmetic with configurable endianness
  - Vector floating point provided via coprocessor
  - Byte addressable memory
  - Virtual memory support
  - Built-in serial port
  - Facilities for kernelized operating system

StrongARM Memory Architecture

- Memory architecture
  - Uses 32-bit linear address space
  - Byte addressable
- Memory Mapping
  - Allocation of address space to different system components
  - Access to memory is translated into access to component
  - Needs to be carefully crafted
- StrongARM assumes byte addressable memory
  - Underlying memory uses different size (SDRAM)
  - How does this work?
- Support for Virtual Memory
  - For demand paging to secondary storage

StrongARM Memory Map



Shared Memory Address Issues

- Memory is shared between StrongARM and Microengines
- Same data, but different addresses
- What impact does this have?
  - Pointers need to be translated
  - Data structures with pointers cannot be shared. Why?

StrongARM Peripherals

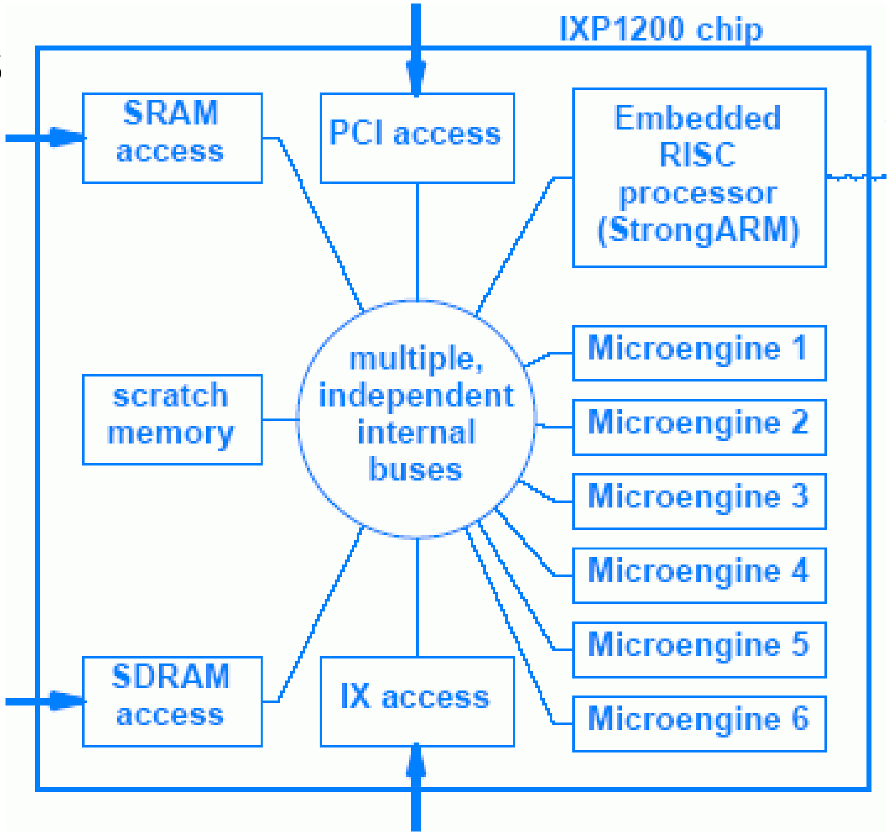
- Peripherals on StrongARM:
- UART
- Four 24-bit countdown timers
  - Can be configured to 1, 1/16, 1/256 of StrongARM clock
- Four general purpose pins
  - For special off-chip devices
- One real-time clock
  - Tick per second
- Clock is for large granularity timing (e.g., route aging), counters are for small granularity

StrongARM Misc

- StrongARM can support kernelized OS
  - Kernel at highest priority
  - Kernel controls I/O and devices
  - User-level processes with lower privileges
- Coprocessor 15
  - MMU configuration
  - Breakpoints for testing
- Summary
  - StrongARM is full-blown processor with powerful and general features

Microengines

- Microengines are data-path processors of IXP1200
- IPX1200 has 6 microengines
- Simpler than StrongARM
- A bit more complex to use
- Often abbeviated as uE



Microengine Functions

- uEs handle ingress and egress packet processing:
  - Packet ingress from physical layer hardware
  - Checksum verification
  - Header processing and classification
  - Packet buffering in memory
  - Table lookup and forwarding
  - Header modification
  - Checksum computation
  - Packet egress to physical layer hardware

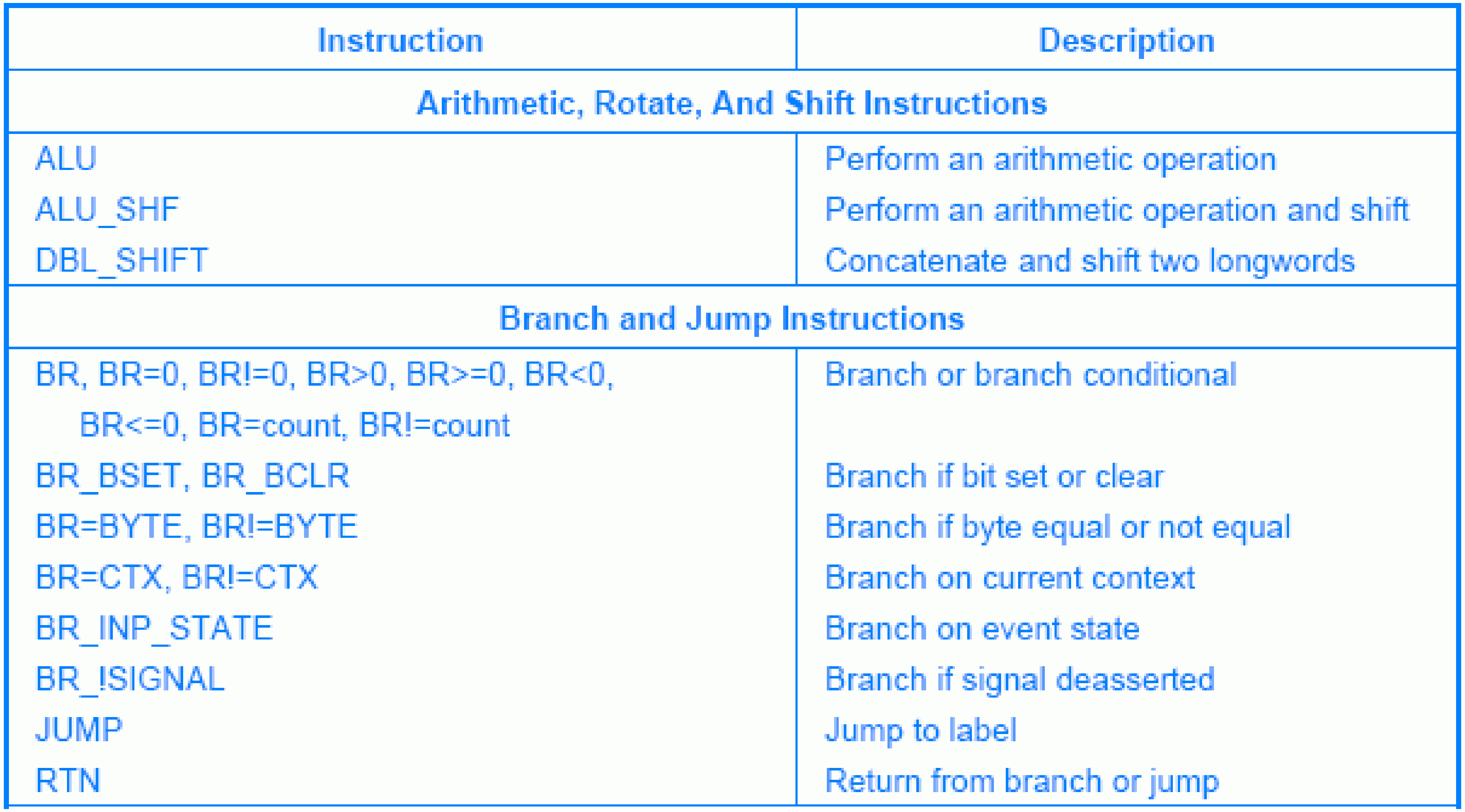
Microengine Architecture

- uE characteristics:
  - Programmable microcontroller
  - RISC design
  - 128 general-purpose registers
  - 128 transfer registers
  - Hardware support for 4 threads and context switching
  - Five-stage execution pipeline
  - Control of an Arithmetic and Logic Unit
  - Direct access to various functional units

uE as Microsequencer

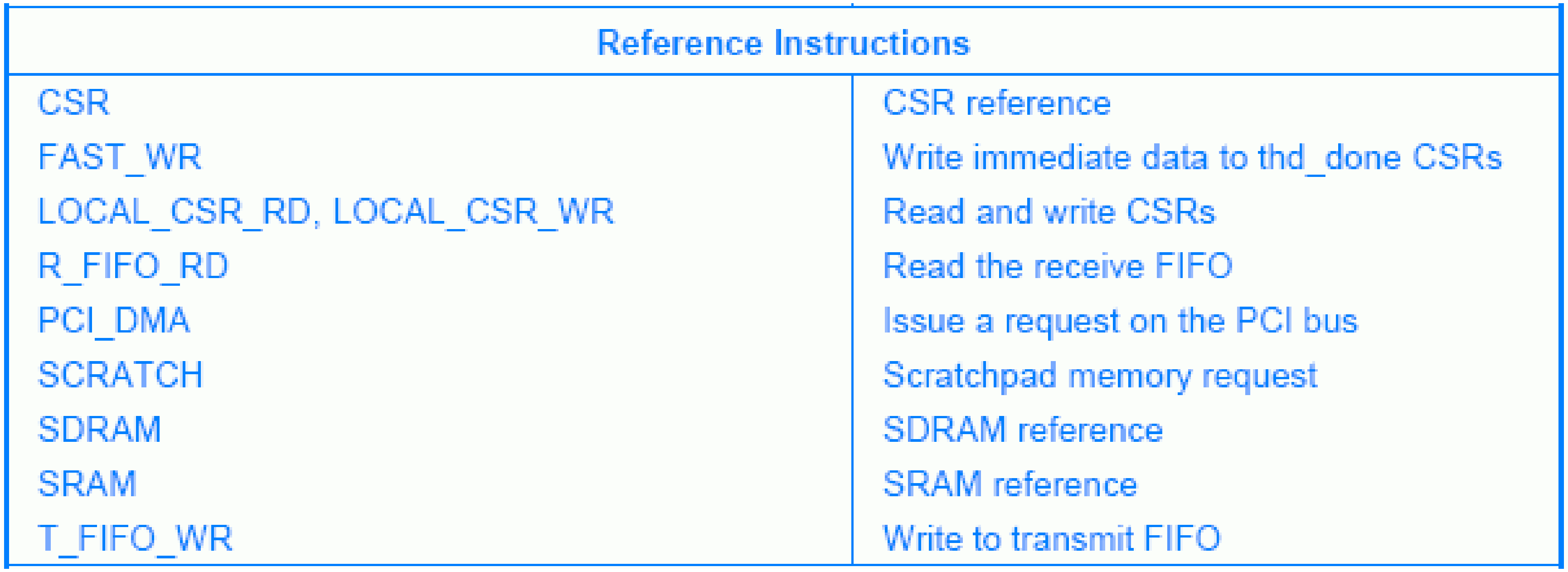
- Microsequencer does not contain native operations
  - Control unit is much “simpler”
- Instead of using instructions, uE invokes functional units
- Example 1:
  - uE does not have ADD R2,R3 instruction
  - Instead: ALU ADD R2, R3
  - “ALU” indicates that ALU should be used
  - “ADD” is a parameter to ALU
- Example 2:
  - Memory access not by simple LOAD R2, 0xdeadbeef
  - Instead: SRAM LOAD R2, 0xdeadbeef
- Altogether similar to normal processor, but more basic

Microengine Instruction Set (1)



Instruction	Description
Arithmetic, Rotate, And Shift Instructions	
ALU	Perform an arithmetic operation
ALU_SHF	Perform an arithmetic operation and shift
DBL_SHIFT	Concatenate and shift two longwords
Branch and Jump Instructions	
BR, BR=0, BR!=0, BR>0, BR>=0, BR<0, BR<=0, BR=count, BR!=count	Branch or branch conditional
BR_BSET, BR_BCLR	Branch if bit set or clear
BR=BYTE, BR!=BYTE	Branch if byte equal or not equal
BR=CTX, BR!=CTX	Branch on current context
BR_INP_STATE	Branch on event state
BR_!SIGNAL	Branch if signal deasserted
JUMP	Jump to label
RTN	Return from branch or jump

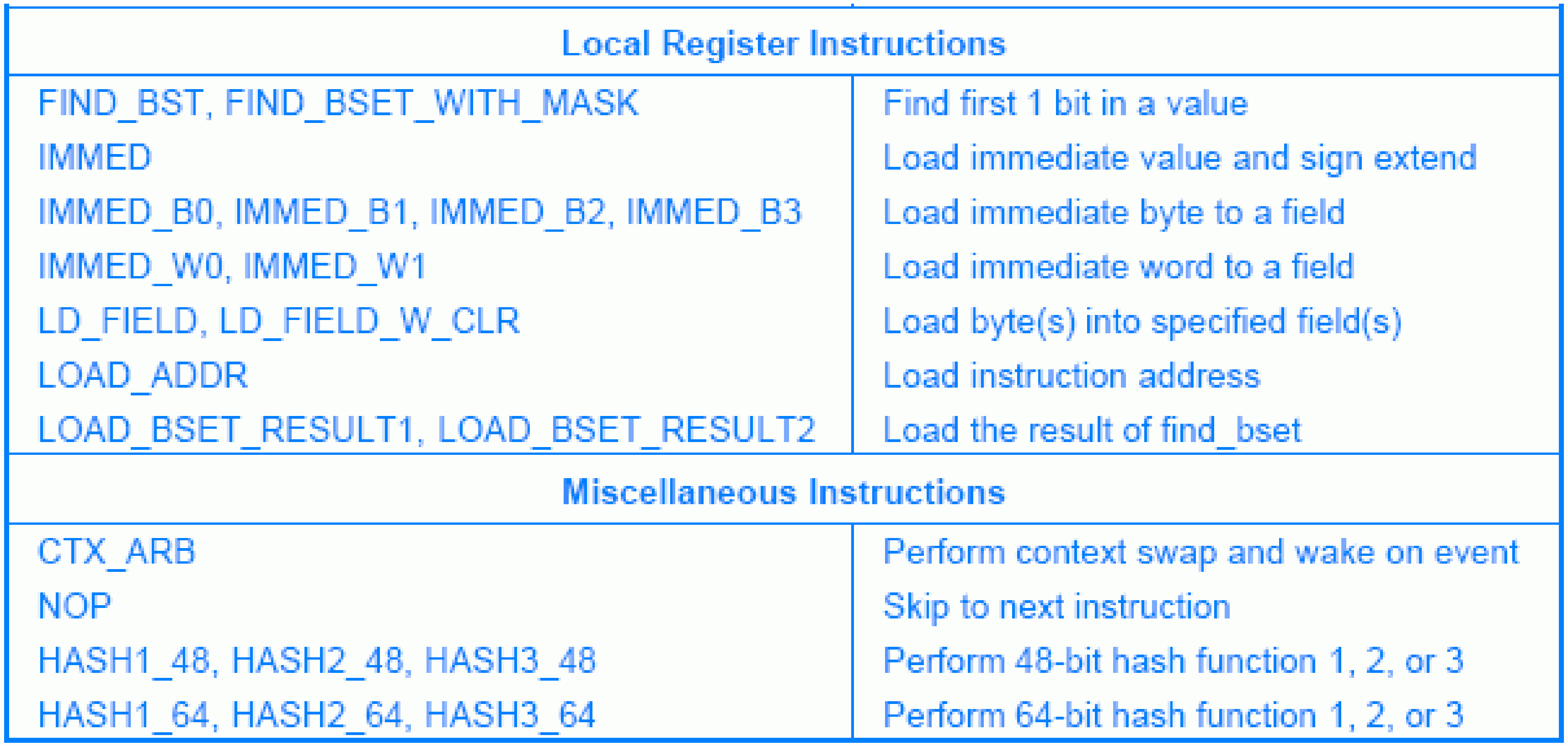
Microengine Instruction Set (2)



Reference Instructions	
CSR	CSR reference
FAST_WR	Write immediate data to thd_done CSRs
LOCAL_CSR_RD, LOCAL_CSR_WR	Read and write CSRs
R_FIFO_RD	Read the receive FIFO
PCI_DMA	Issue a request on the PCI bus
SCRATCH	Scratchpad memory request
SDRAM	SDRAM reference
SRAM	SRAM reference
T_FIFO_WR	Write to transmit FIFO

- CSR = Control and Status Register

Microengine Instruction Set (3)



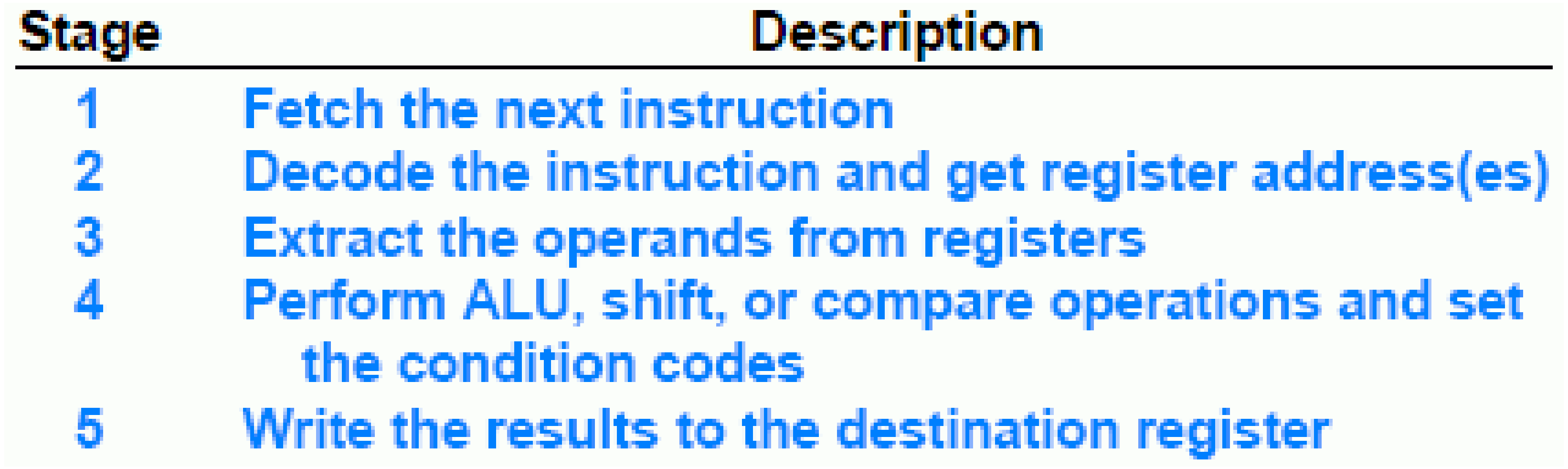
Local Register Instructions	
FIND_BST, FIND_BSET_WITH_MASK	Find first 1 bit in a value
IMMED	Load immediate value and sign extend
IMMED_B0, IMMED_B1, IMMED_B2, IMMED_B3	Load immediate byte to a field
IMMED_W0, IMMED_W1	Load immediate word to a field
LD_FIELD, LD_FIELD_W_CLR	Load byte(s) into specified field(s)
LOAD_ADDR	Load instruction address
LOAD_BSET_RESULT1, LOAD_BSET_RESULT2	Load the result of find_bset
Miscellaneous Instructions	
CTX_ARB	Perform context swap and wake on event
NOP	Skip to next instruction
HASH1_48, HASH2_48, HASH3_48	Perform 48-bit hash function 1, 2, or 3
HASH1_64, HASH2_64, HASH3_64	Perform 64-bit hash function 1, 2, or 3

Microengine Memories

- uEs views memories separately
  - Not one address space like StrongARM
- Requires programmer to decide on memories to use
  - Different memories require different instructions
- Also: instruction store is in different memory than data
  - Not a van-Neumann/Princeton architecture…

Execution Pipeline

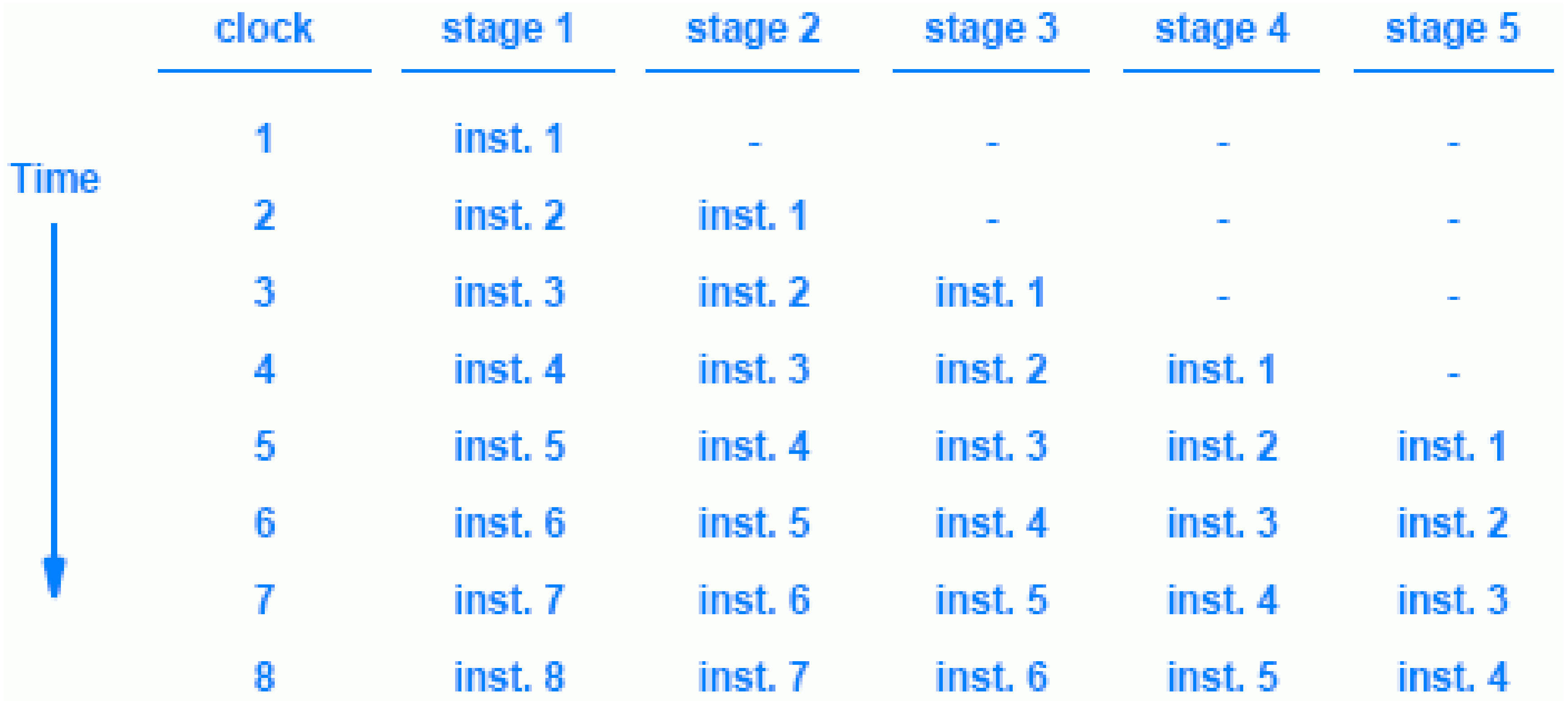
- uEs have five-stage pipeline:



Stage	Description
1	Fetch the next instruction
2	Decode the instruction and get register address(es)
3	Extract the operands from registers
4	Perform ALU, shift, or compare operations and set the condition codes
5	Write the results to the destination register

- In proper pipeline operation, one instruction is executed per cycle

Pipelining



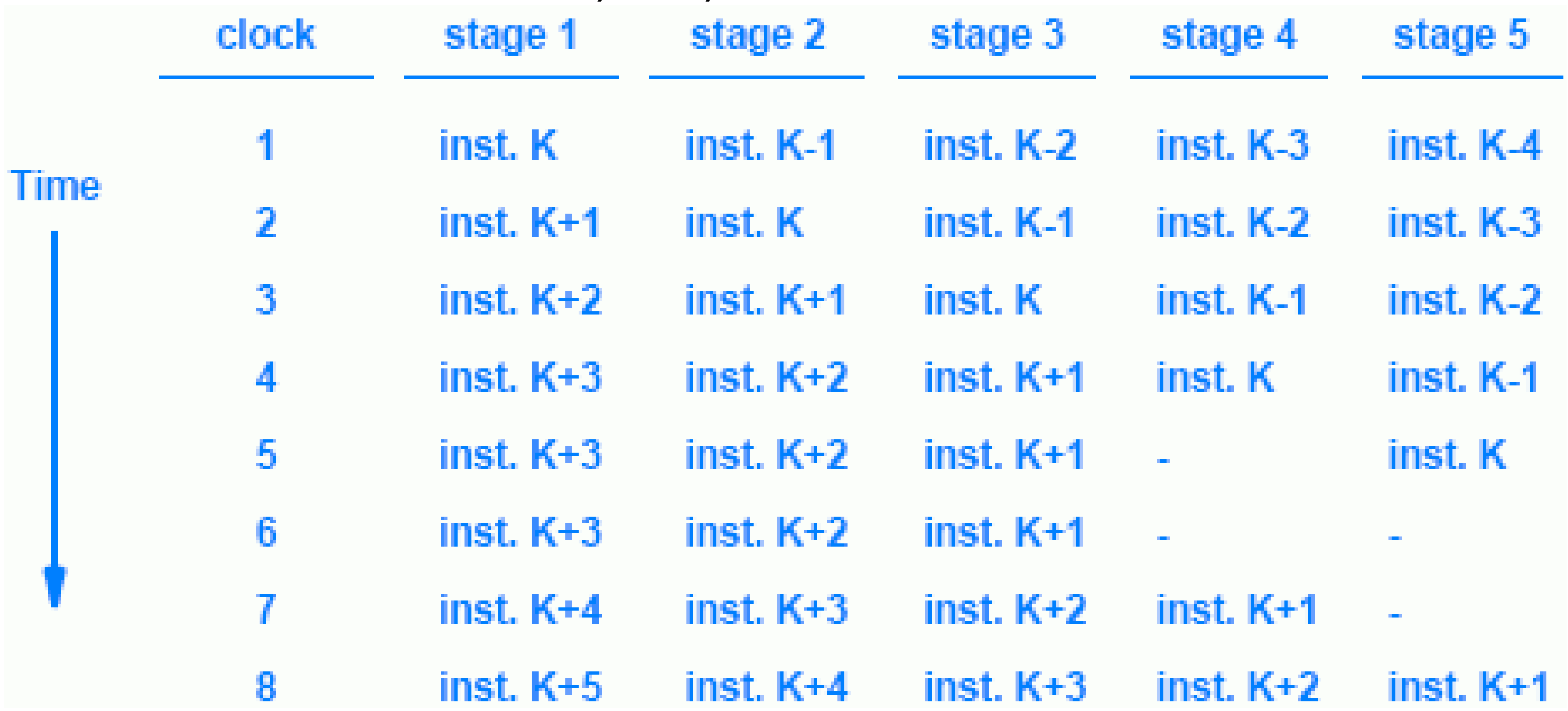
clock	stage 1	stage 2	stage 3	stage 4	stage 5
1	inst. 1	-	-	-	-
2	inst. 2	inst. 1	-	-	-
3	inst. 3	inst. 2	inst. 1	-	-
4	inst. 4	inst. 3	inst. 2	inst. 1	-
5	inst. 5	inst. 4	inst. 3	inst. 2	inst. 1
6	inst. 6	inst. 5	inst. 4	inst. 3	inst. 2
7	inst. 7	inst. 6	inst. 5	inst. 4	inst. 3
8	inst. 8	inst. 7	inst. 6	inst. 5	inst. 4

Pipelining Problems

- What can lead to cases where pipeline does not operate as desired?
  - Data dependencies
  - Control dependencies
  - Memory accesses
- What happens in either case?
- How can these cases be made less frequent?
- How can the impact be reduced?

Pipeline Stalls

- K: ADD R2, R1, R2
- K+1: ADD R3, R2, R3



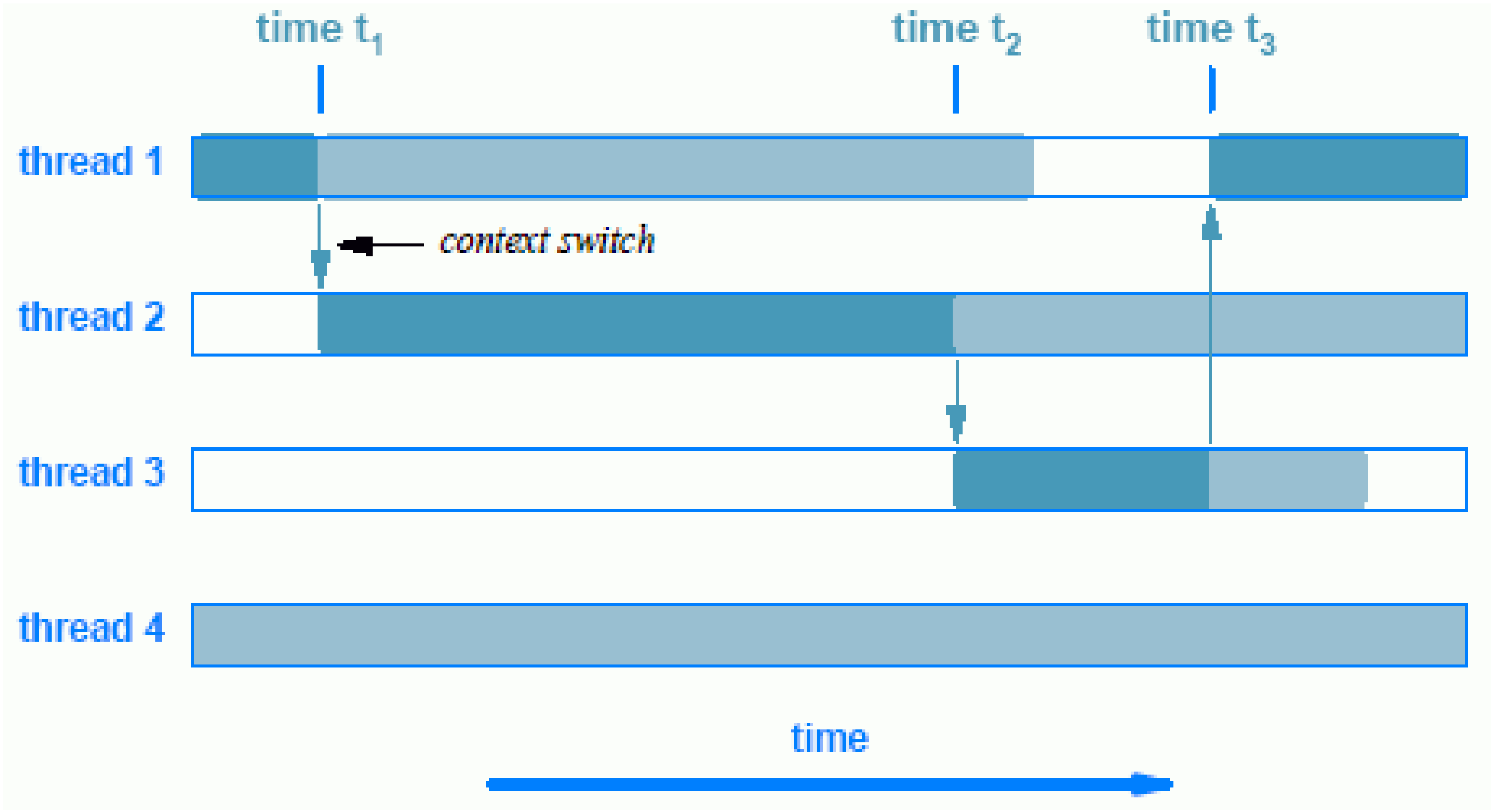
clock	stage 1	stage 2	stage 3	stage 4	stage 5
1	inst. K	inst. K-1	inst. K-2	inst. K-3	inst. K-4
2	inst. K+1	inst. K	inst. K-1	inst. K-2	inst. K-3
3	inst. K+2	inst. K+1	inst. K	inst. K-1	inst. K-2
4	inst. K+3	inst. K+2	inst. K+1	inst. K	inst. K-1
5	inst. K+3	inst. K+2	inst. K+1	-	inst. K
6	inst. K+3	inst. K+2	inst. K+1	-	-
7	inst. K+4	inst. K+3	inst. K+2	inst. K+1	-
8	inst. K+5	inst. K+4	inst. K+3	inst. K+2	inst. K+1

- Control dependencies, memory have even bigger impact

Hardware Threads

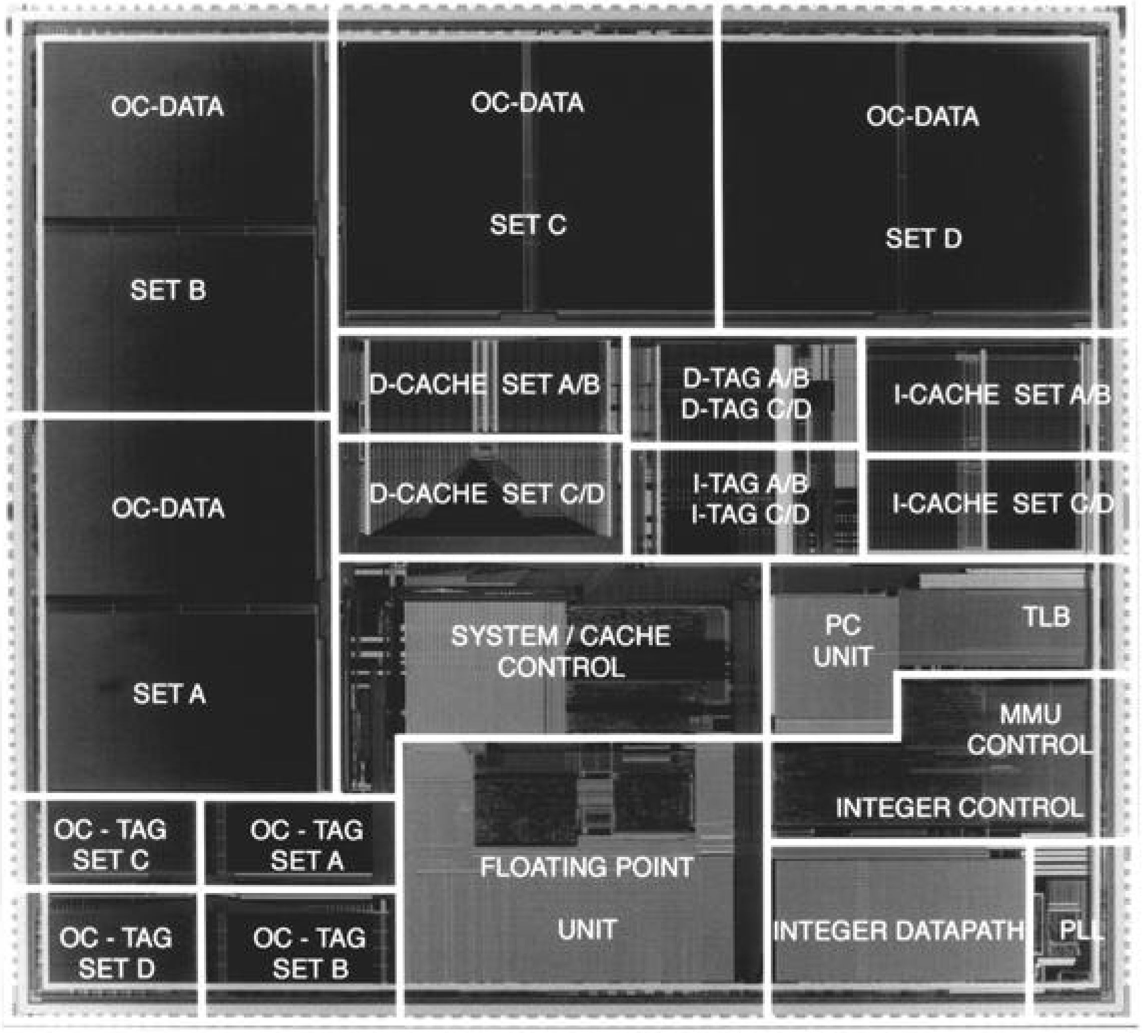
- uEs support four hardware thread contexts
  - One thread can execute at any given time
  - When stall occurs, uE can switch to other thread (if not stalled)
- Very low overhead for context switch
  - “Zero-cycle context switch”
  - Effectively can take around three cycles due to pipeline flush
- Switching rules
  - If thread stalls, check if next is ready for processing
  - Keep trying until ready thread is found
  - If none is available, stall uE and wait for any thread to unblock
- Improves overall throughput
- Side note: why not have 24 uEs with 1 thread?

Threading Illustration



Processor Component Proportions

- "Random" RISC processor (MIPS R7000)
- 300 MHz, 16k/16k caches, .25 um, 1997
- Memory takes most area



Next Class

- Continue with Microengines
  - Instruction store, hardware registers
  - FBI and FIFO
  - Hash unit
- SDK
- Read chapters 20 & 21