Cortex-M0+ CPU Core

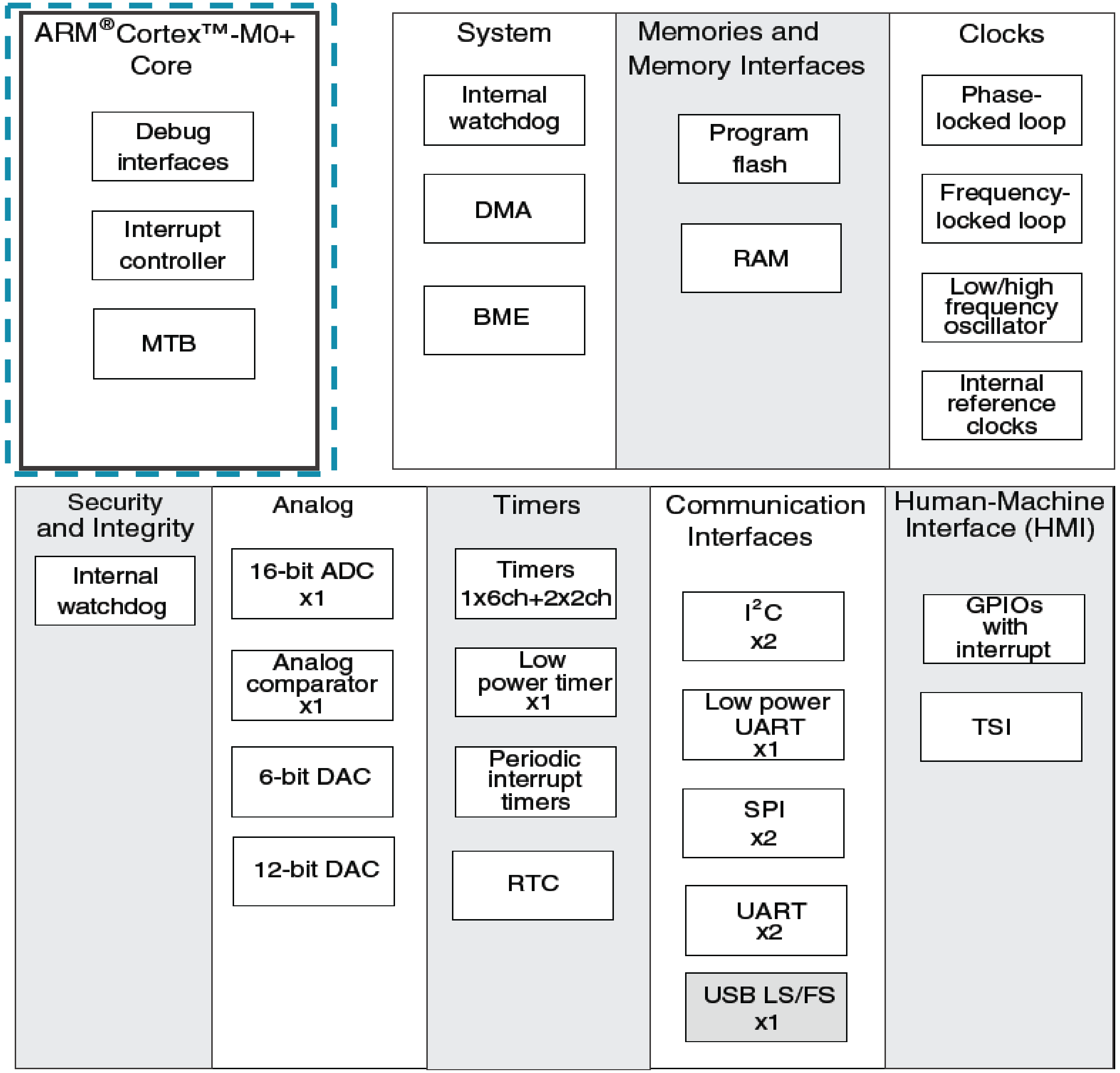
Ravi Suppiah
Lecturer, NUS SoC

Overview

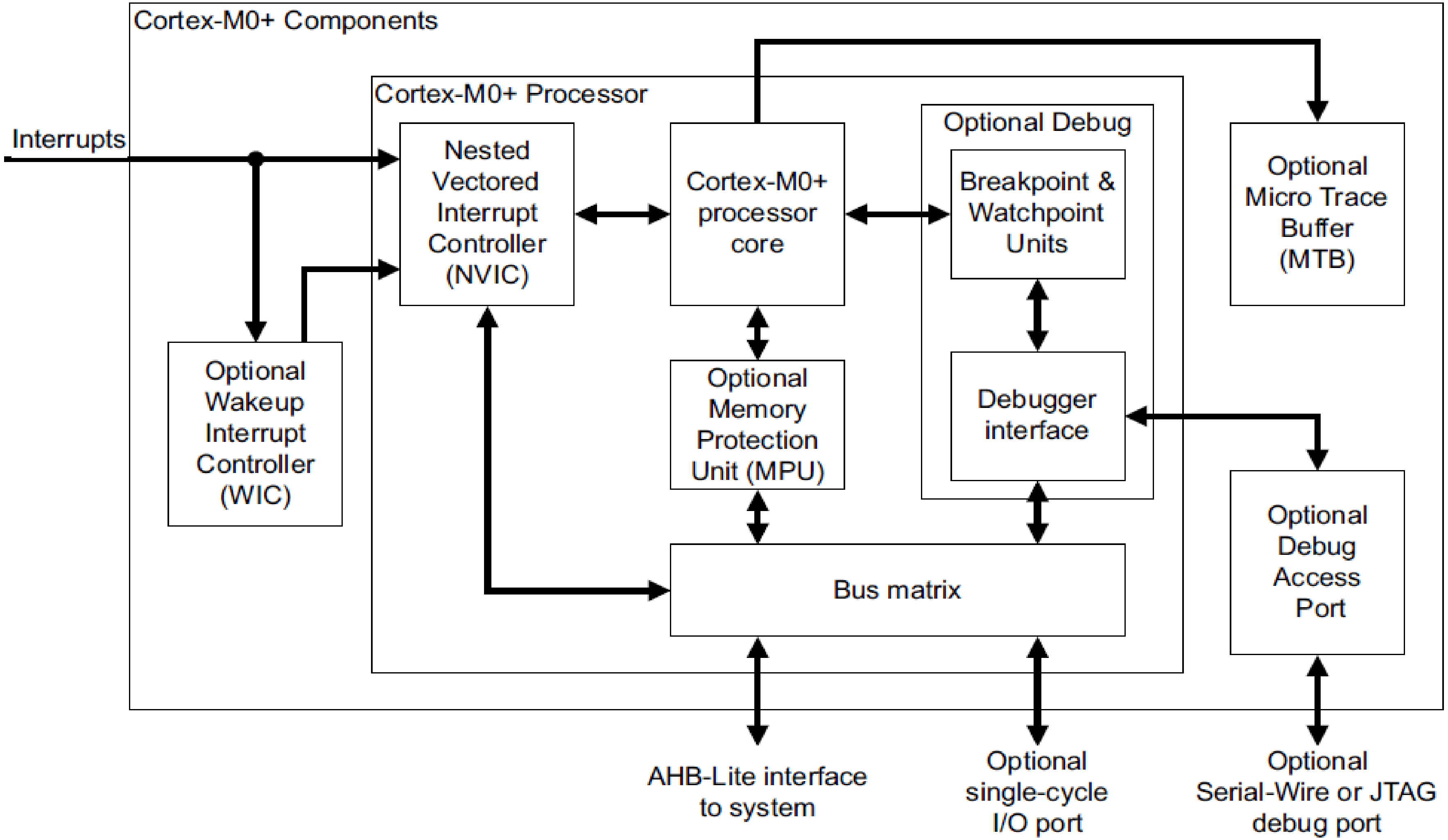
- Cortex-M0+ Processor Core Registers
- Memory System and Addressing
- Thumb Instruction Set
- References
  - DDI0419C Architecture ARMv6-M Reference Manual

Microcontroller vs. Microprocessor

- Both have a CPU core to execute instructions
- Microcontroller has peripherals for embedded interfacing and control
  - Analog
  - Non-logic level signals
  - Timing
  - Clock generators
  - Communications
    - point to point
    - network
  - Reliability and safety



Cortex-M0+ Core

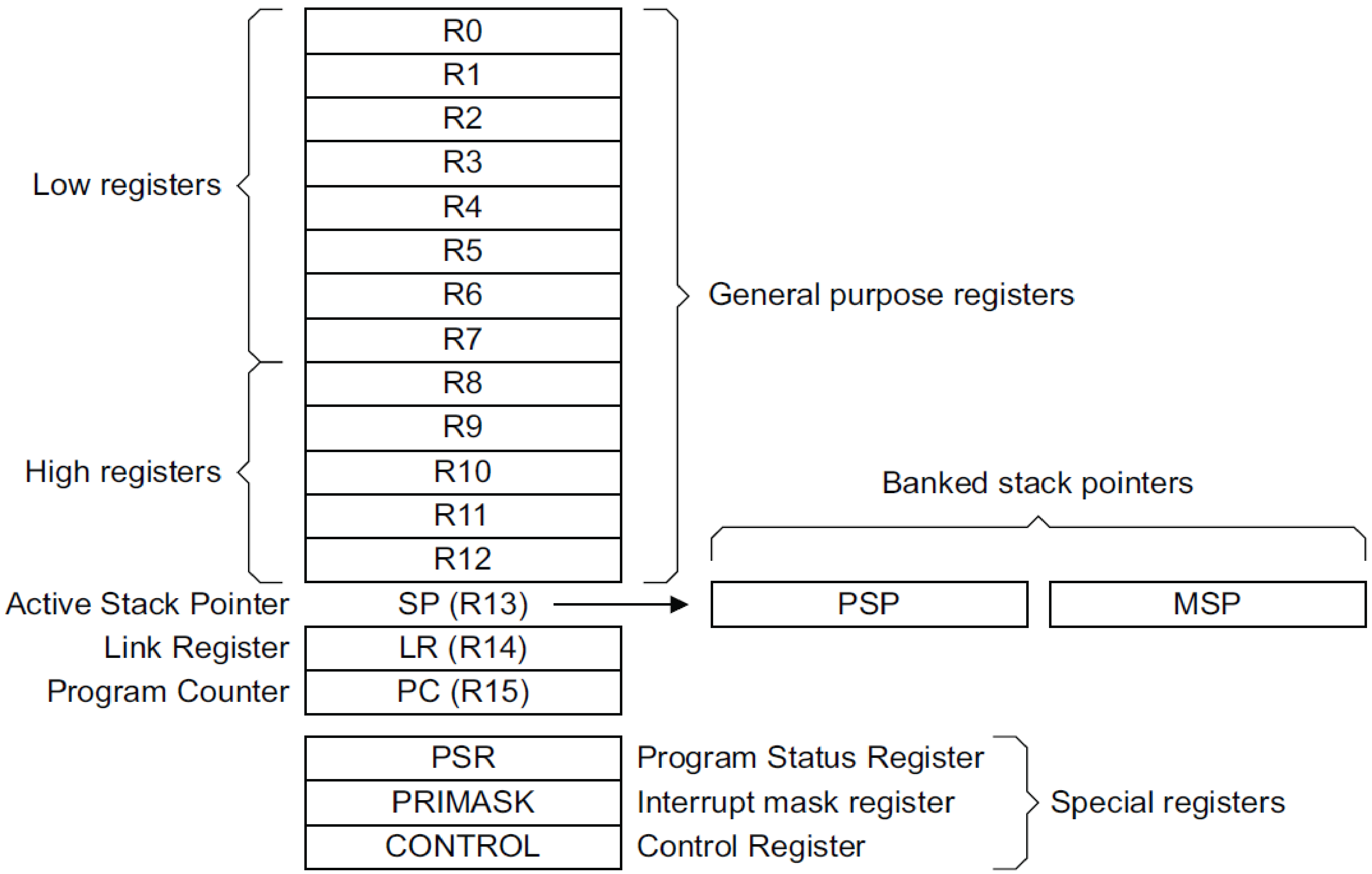


Architectures and Memory Speed

- Load/Store Architecture
  - Developed to simplify CPU design and improve performance
    - *Memory wall*: CPUs keep getting faster than memory
    - Memory accesses slow down CPU, limit compiler optimizations
    - Change instruction set to make most instructions *independent* of memory
  - Data processing instructions can access registers only
    1. Load data into the registers
    2. Process the data
    3. Store results back into memory
  - More effective when more registers are available

- Register/Memory Architecture
  - Data processing instructions can access memory or registers
  - Memory wall is not very high at lower CPU speeds (e.g. under 50 MHz)

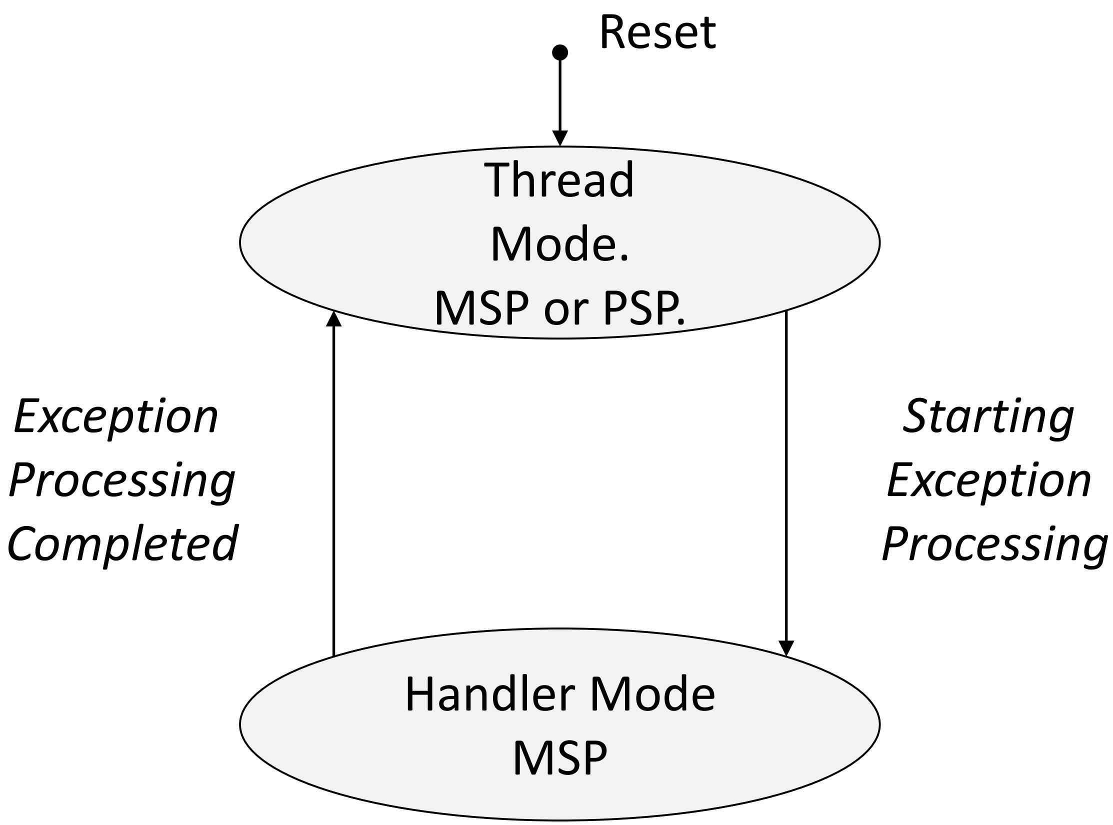
ARM Processor Core Registers



ARM Processor Core Registers (32 bits each)

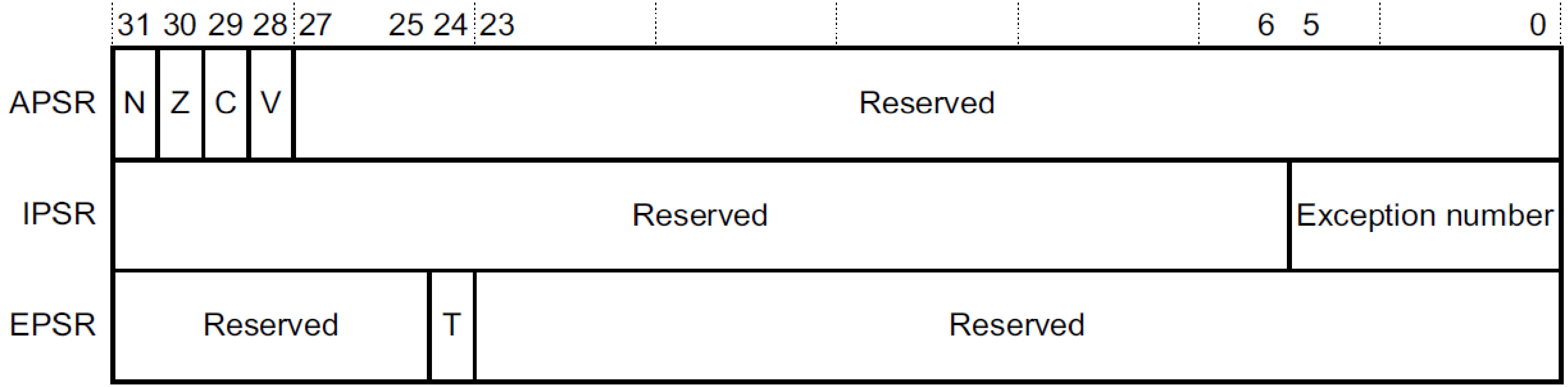
- R0-R12 - General purpose registers for data processing
- SP - Stack pointer (R13)
  - Can refer to one of two SPs
    - Main Stack Pointer (MSP)
    - Process Stack Pointer (PSP)
  - Uses MSP initially, and whenever in Handler mode
  - When in Thread mode, can select either MSP or PSP using SPSEL flag in CONTROL register.
- LR - Link Register (R14)
  - Holds return address when called with Branch & Link instruction (B&L)
- PC - program counter (R15)

Operating Modes



- Which SP is active depends on operating mode, and SPSEL (CONTROL register bit 1)
  - SPSEL == 0: MSP
  - SPSEL == 1: PSP

ARM Processor Core Registers



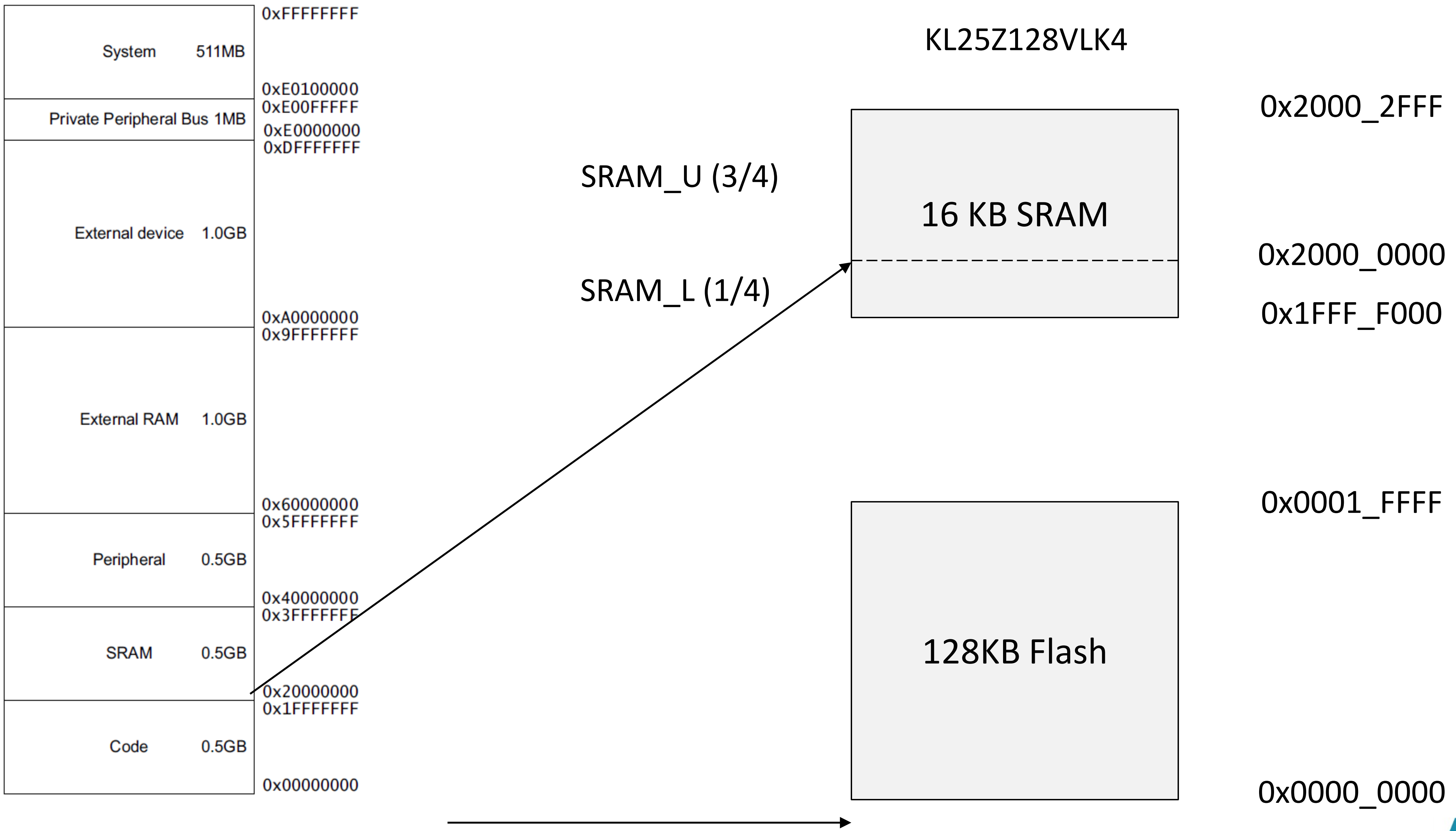
- Program Status Register (PSR) is three views of same register
  - Application PSR (APSR)
    - Condition code flag bits Negative, Zero, oVerflow, Carry
  - Interrupt PSR (IPSR)
    - Holds exception number of currently executing ISR
  - Execution PSR (EPSR)
    - Thumb state

ARM Processor Core Registers

- PRIMASK - Exception mask register
  - Bit 0: PM Flag
    - Set to 1 to prevent activation of all exceptions with configurable priority
  - Access using CPS, MSR and MRS instructions
  - Use to prevent data race conditions with code needing atomicity

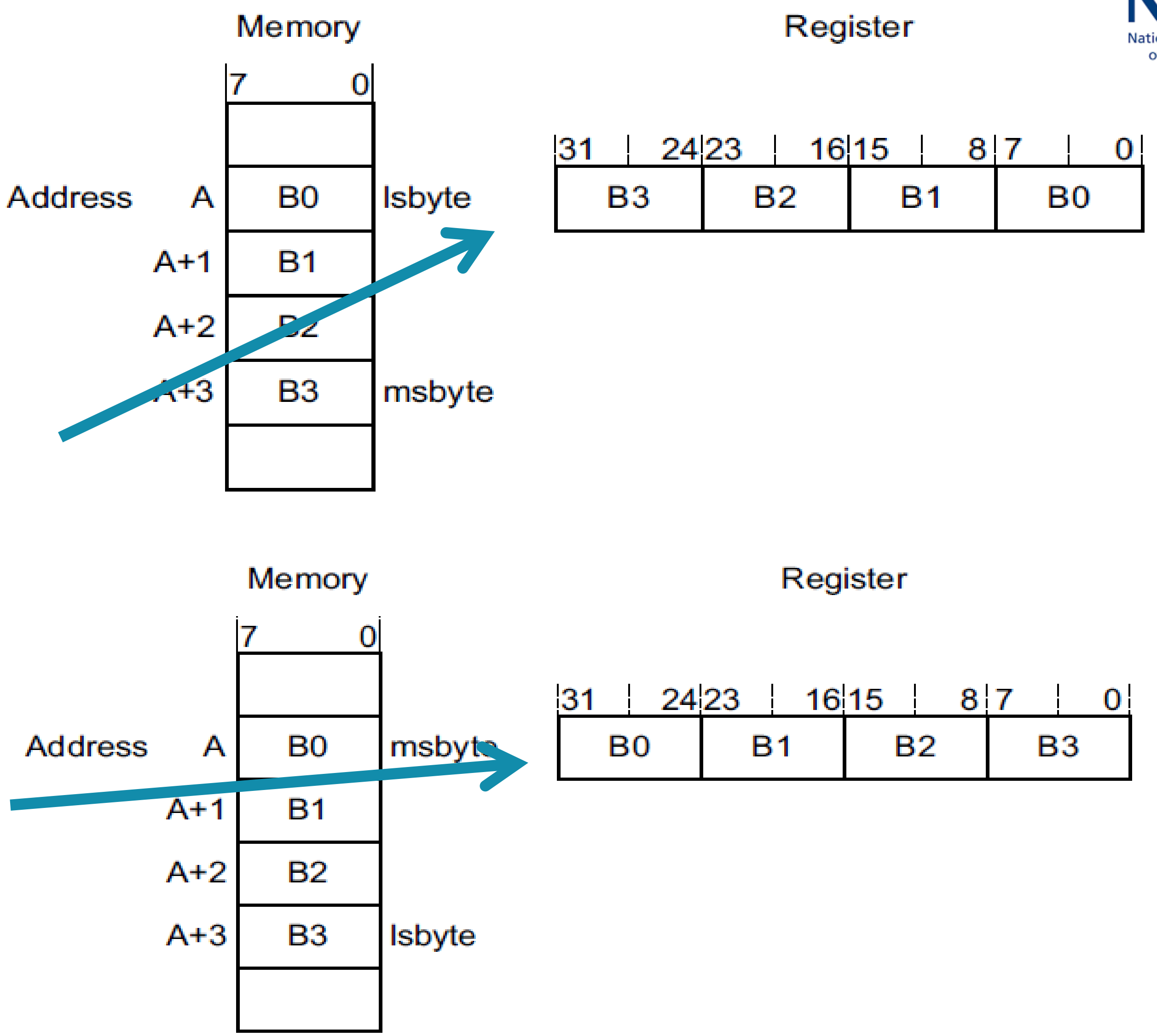
- CONTROL
  - Bit 1: SPSEL flag
    - Selects SP when in thread mode: MSP (0) or PSP (1)
  - Bit 0: nPRIV flag
    - Defines whether thread mode is privileged (0) or unprivileged (1)
  - With OS environment,
    - Threads use PSP
    - OS and exception handlers (ISRs) use MSP

Memory Maps For Cortex M0+ and MCU



Endianness

- For a multi-byte value, in what order are the bytes stored?
- Little-Endian: Start with least-significant byte
- Big-Endian: Start with most-significant byte



ARMv6-M Endianness

- Instructions are always little-endian
- Loads and stores to Private Peripheral Bus are always little-endian
- Data: Depends on implementation, or from reset configuration
  - Kinetis processors are little-endian

ARM, Thumb and Thumb-2 Instructions

- ARM instructions optimized for resource-rich high-performance computing systems
  - Deeply pipelined processor, high clock rate, wide (e.g. 32-bit) memory bus
- Low-end embedded computing systems are different
  - Slower clock rates, shallow pipelines
  - Different cost factors – e.g. code size matters much more, bit and byte operations critical
- Modifications to ARM ISA to fit low-end embedded computing
  - 1995: Thumb instruction set
    - 16-bit instructions
    - Reduces memory requirements (and performance slightly)
  - 2003: Thumb-2 instruction set
    - Adds some 32 bit instructions
    - Improves speed with little memory overhead
  - CPU decodes instructions based on whether in Thumb state or ARM state - controlled by T bit

Instruction Set

- Cortex-M0+ core implements ARMv6-M Thumb instructions
- Only uses Thumb instructions, always in Thumb state
  - Most instructions are 16 bits long, some are 32 bits
  - Most 16-bit instructions can only access low registers (R0-R7), but some can access high registers (R8-R15)
- Thumb state indicated by program counter being odd (LSB = 1)
  - Branching to an even address will cause an exception, since switching back to ARM state is not allowed
- Conditional execution only supported for 16-bit branch
- 32 bit address space
- Half-word aligned instructions
- See ARMv6-M Architecture Reference Manual for specifics per instruction (Section A.6.7)

Thank You!

- Lets get some I/O done.