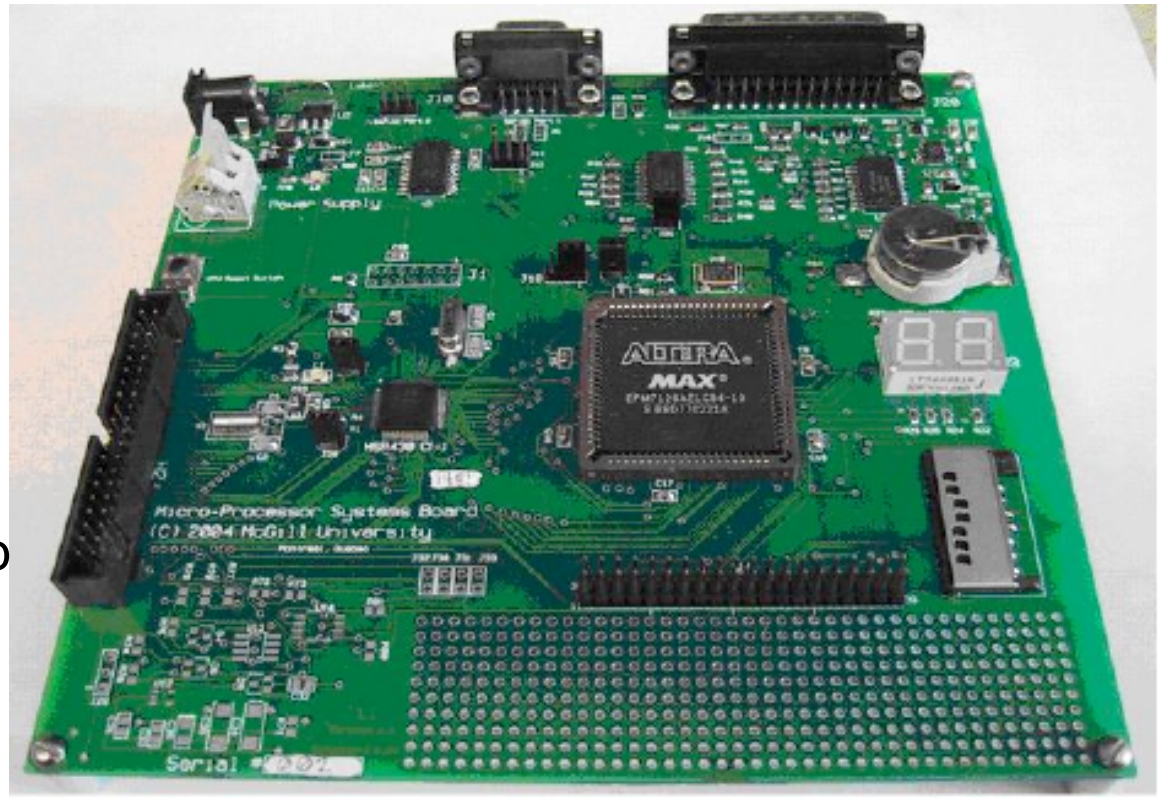
# ECSE 426
# ISA, Assemblers and Labs

Zeljko Zilic

Room 546
McConnell Building
zeljko@ece.mcgill.ca
www.macs.ece.mcgill.ca/~zeljko




McGill

McGill

# Course Organization

- **Instructor:** Prof. Zeljko Zilic, Rm. 546, McConnell
  398-1834, Fax: 398-4470, MyCourses
  e-mail: zeljko@ece.mcgill.ca
- **Office Hours**: Wed. 11:30-12:30; by appointment; e-mail (use ECSE444 in subject line)
- **Tutorials**: As announced throughout semester
- **Lab Demoing**: will schedule by online tools
- **Manned Lab (TAs)**: As per calendar in MyCourses
- **Quizzes**: Will announce, stay tuned

# Course Content

- Top-down approach to microprocessor programming and design
- Lectures focus on structured computer organization, and progress through "layers" :
  - Problem-oriented language (embedded C) + assembly language
  - Instruction set architecture
  - Microarchitecture
  - Hardware
- Introduction of design principles and impact on real-world architectures
  - ARM Cortex M4 Architecture and ARM Cortex M family in general

Acknowledgements: Hamacher, Vranesic, Zaky, Manjikian for "Computer Organization and Emb. Systems". [HVZM11]

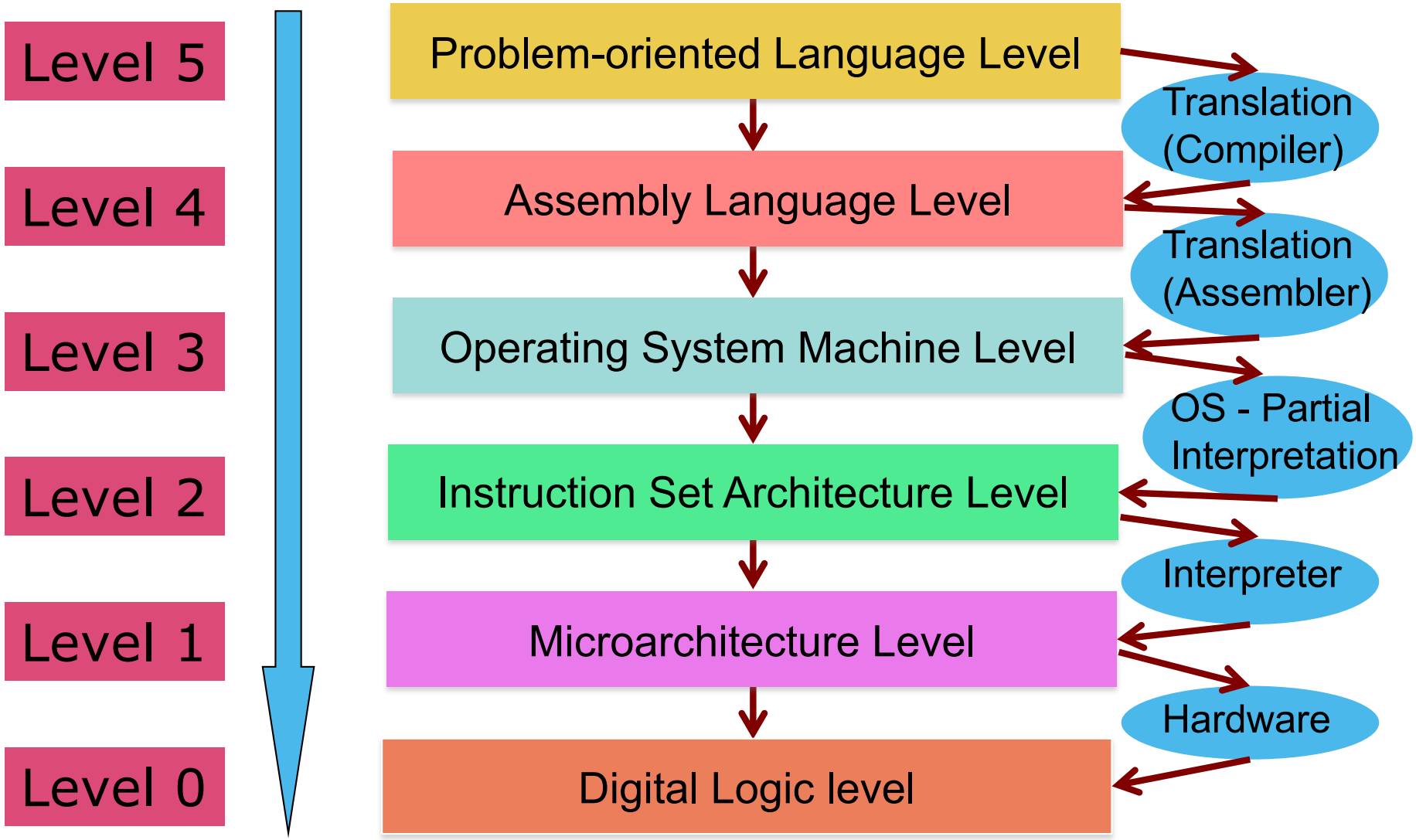
# Course Objectives Rephrased

- Understand microprocessor-based systems
- Get familiar with basic tools
- Skills in machine interfacing, assembler and embedded C programming
- Design a sizeable embedded system
  - Previous projects: Music player, file swapping system, PDAs (with handwriting recognition), wireless data collection systems
- Build teamwork skills

# Today's Lecture

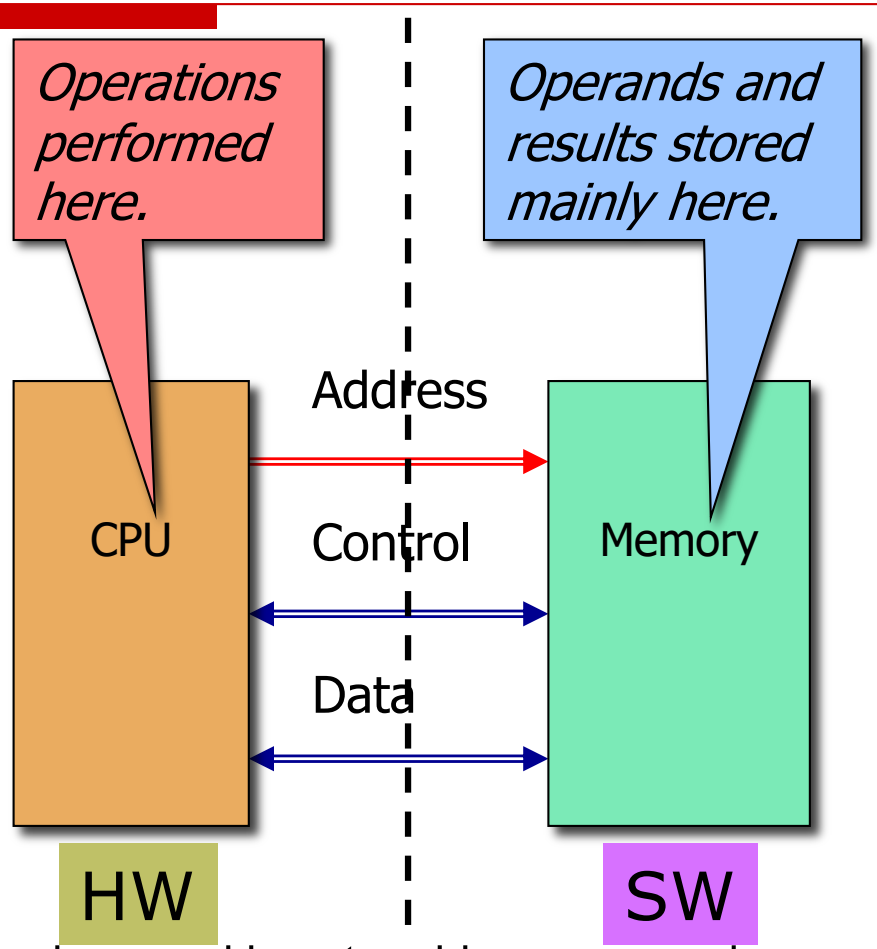
- ISA
  - Processor Resources
  - Instruction Types
  - Addressing Modes
- Assembly Language and Tutorial Complement
  - Introduction to the ARM Cortex Assembler
  - Lab 1 Overview and Hints

# Contemporary Multilevel Machines



# Instruction Set Architecture

- Interface between HW and SW
  - Virtual/Real Machine
    - Many possible implementations of ISA
- Given by
  - Resources
    - Processor Registers
    - Execution Units
  - Operations
    - Instruction Types
    - Data Types
    - Addressing Modes




i.e., where and how to address operands

# ARM Processor – Brief History

- ARM was developed as Acorn Computers Ltd., Cambridge, England (1983-1985)

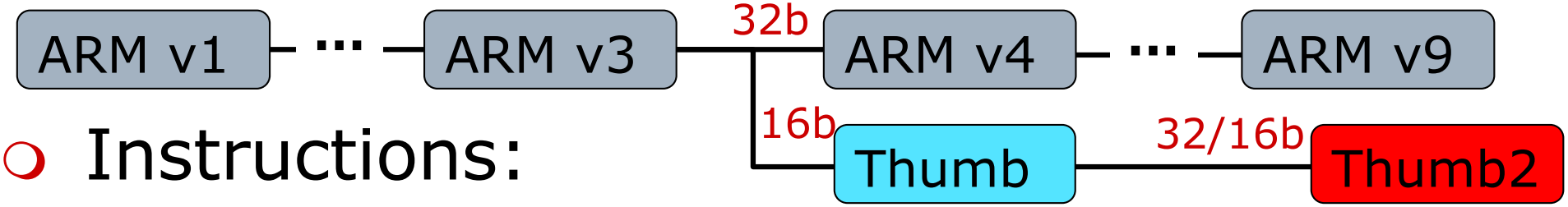

- RISC processor concept was introduced in 1980s
- Acorn RISC Machine -> Advanced RISC Machines
  - ARM Ltd. - 1990

# ARM Processor - Architecture

- A 32-bit Enhanced RISC processor with register-to-register, three operand instruction set
  - All operands are 32-bit wide
- Employs Load/Store Architecture (RISC, really)
  - Operations operate on registers and not in memory locations

# ARM Cortex ISA: ARMv7-M

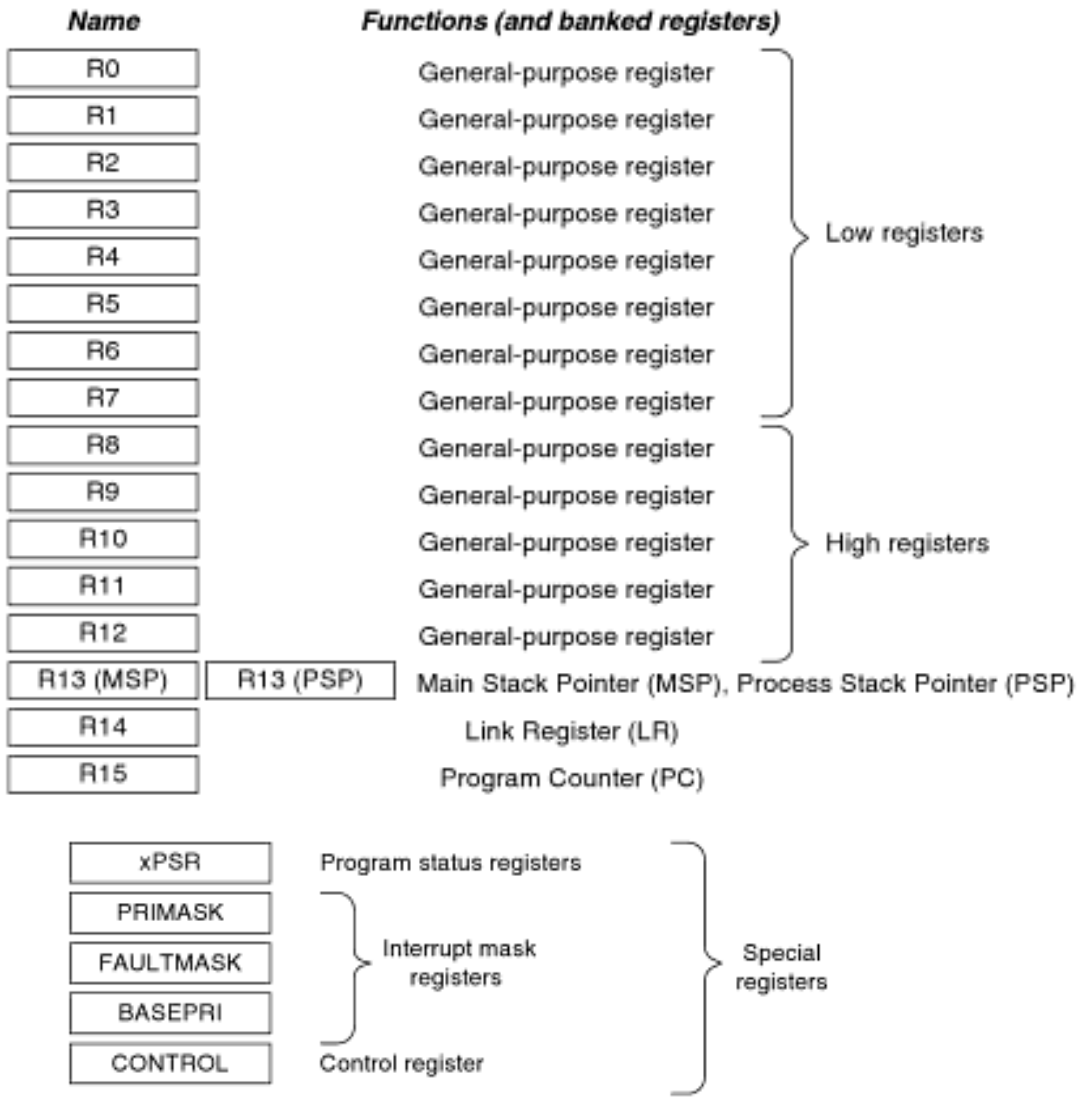
- Implements Thumb2 specification




- Instructions:
  - Data movement: single and multiple word
  - Data processing: arithmetic, logic, shift, rotate
  - Branches: conditional, subroutine, table (case)
  - State change: int. enable, spec. reg., ITE, SVC
  - Semaphores, Barrier, Hint, Prefetch
  - Coprocessor, floating-point, misc.

# ARM Registers

- Large register set
  - 16 32-bit general purpose registers
  - Program Counter (R15)
  - Link Register (R14)
  - Stack Pointer (R13)
- Registers R0-R3 hold first 4 words of incoming argument



# ARM Register Set Summary

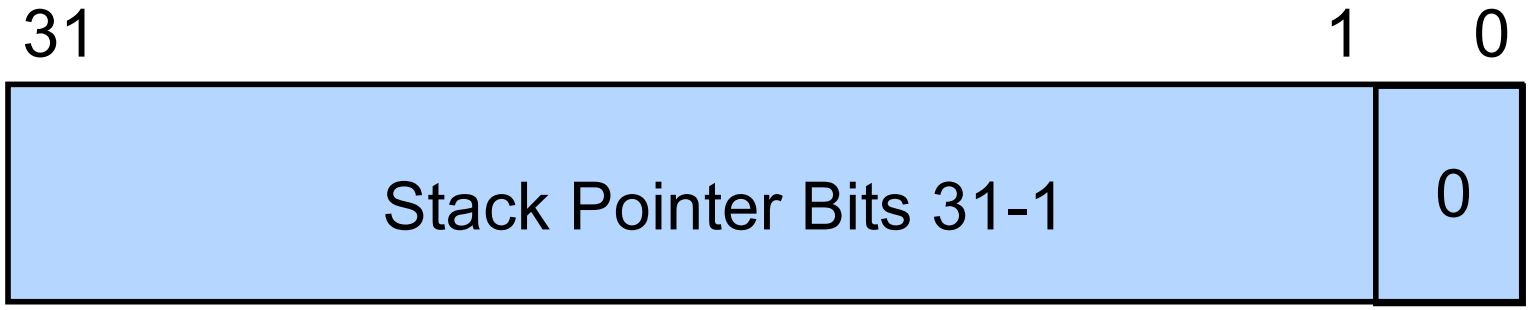
- 16 accessible 32-bit registers (R0-R15) at any time
  - R15 acts as Program Counter (PC)
  - R14 (Link Register) stores subroutine return address
- You can write in assembly language (case in-sensitive):
  - PC for R15,
  - LR for R14,
  - SP for R13
- Current Program Status register (CPSR) that holds conditional codes
- Some registers are not unique as processor exceptions create new instances of R13 and R14
- As return address is not necessarily saved on the stack by a subroutine call, ARM is fast at implementing subroutine return calls

# Program Counter – R15

- Tells you where you are in the program
  - Next instruction address (pipelining blurs that)
- 32 bits, aligned to even addresses
  - LDR PC, =LABEL  Branch to address in LABEL
  - MOV PC, R4  Branch to address contained in R4
  - MOV R0, PC  Not recommended, processor-dependent

# Stack Pointer

- Used by CPU to store return addresses of calls and interrupts
- Don't push and pop the SP and PC !
- SP can also be used by software.
- Initialized into RAM by user, aligned to even addresses



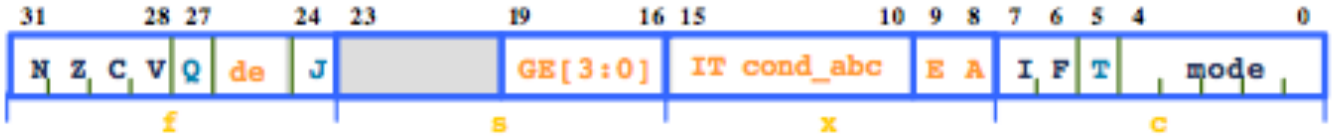
# Current Program Status Register (CPSR)

- CPSR stores the status of current execution
- ARM has more than one CPSR
- In normal mode of operation CPSR contains current values of conditional code bits N, Z, C, and V, and 8 system status bits
- When interrupt occurs, the ARM saves pre-exception values of CPSR in a stored Saved Program Status Register (SPSR)
  - There is one SPSR register for each of the interrupt mode
- SPSR keeps the status of program and processor

# CPSR


- CPSR contains number of flags that report and control operation of CPU
- Condition code flags:
  - N – Negative Result from ALU
  - Z – Zero result from ALU
  - C – ALU operation carried out (carry from ALU operation)
  - V – ALU operation overflown
- 8 lower bits I, F, T anf M[4:0]
  - Interrupt Enable Bits
    - I = 1 - IRQ, interrupt prohibition
    - F =1 - FIQ interrupt prohibition
- T flag reflects the processor running
  - If T = 0 then processor in ARM Mode
  - If T = 1 then processor in THUMB Mode
- M[4:0] – operating mode bits
  - Determine the processor operating mode
- Bits 27-8 are reserved and when conditions change in the code PSR flag or control bits reserve bits do not change

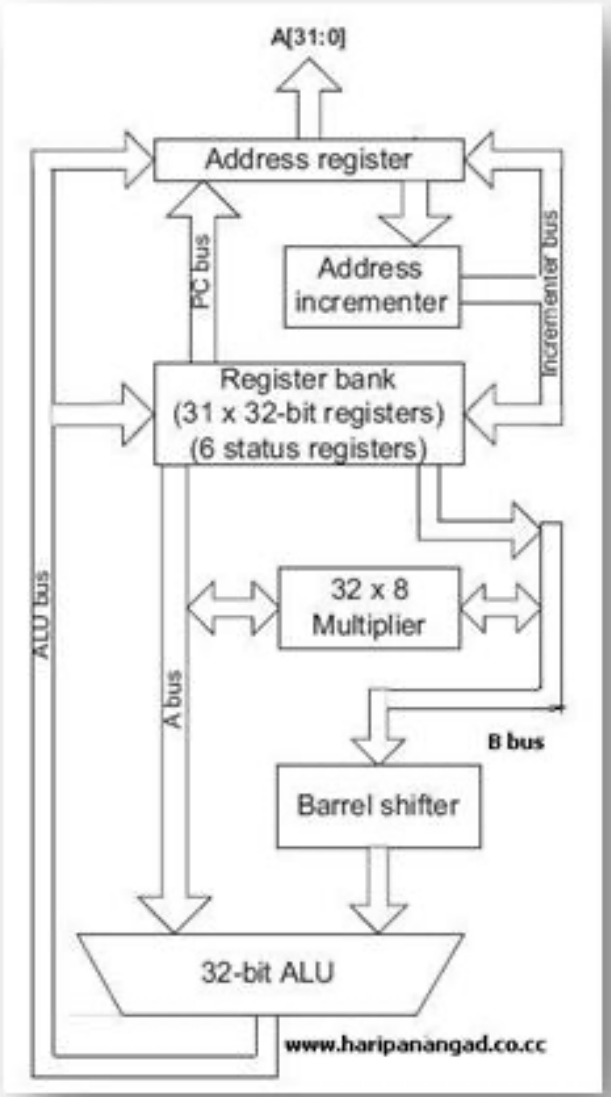
# Program Status Register - Details




- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation oVerflowed
- Sticky Overflow flag - Q flag
  - Architecture 5TE and later only
  - Indicates if saturation has occurred
- J bit
  - Architecture 5TEJ and later only
  - J = 1: Processor in Jazelle state
- Interrupt Disable bits
  - I = 1: Disables IRQ
  - F = 1: Disables FIQ
- T Bit
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
  - Introduced in Architecture 4T
- Mode bits
  - Specify the processor mode
- New bits in V6
  - GE[3:0] used by some SIMD instructions
  - E bit controls load/store endianness
  - A bit disables imprecise data aborts
  - IT [abcde] IF THEN conditional execution of Thumb2 instruction groups

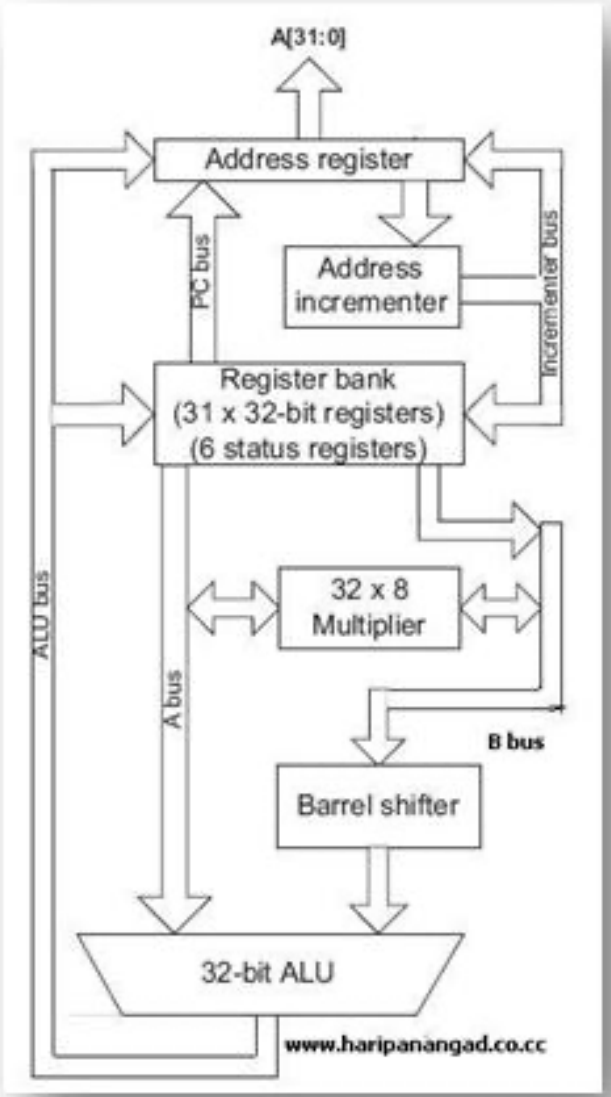
# ARM – Core Datapath

- Separate control and datapath
- In datapath operands are not fetched directly from memory locations
  - Data is placed in register files
  - No data processing takes place in memory
- Instructions typically use 3 registers
  - 2 source registers and 1 destination register
- Barrel shifter preprocesses data before it enters ALU



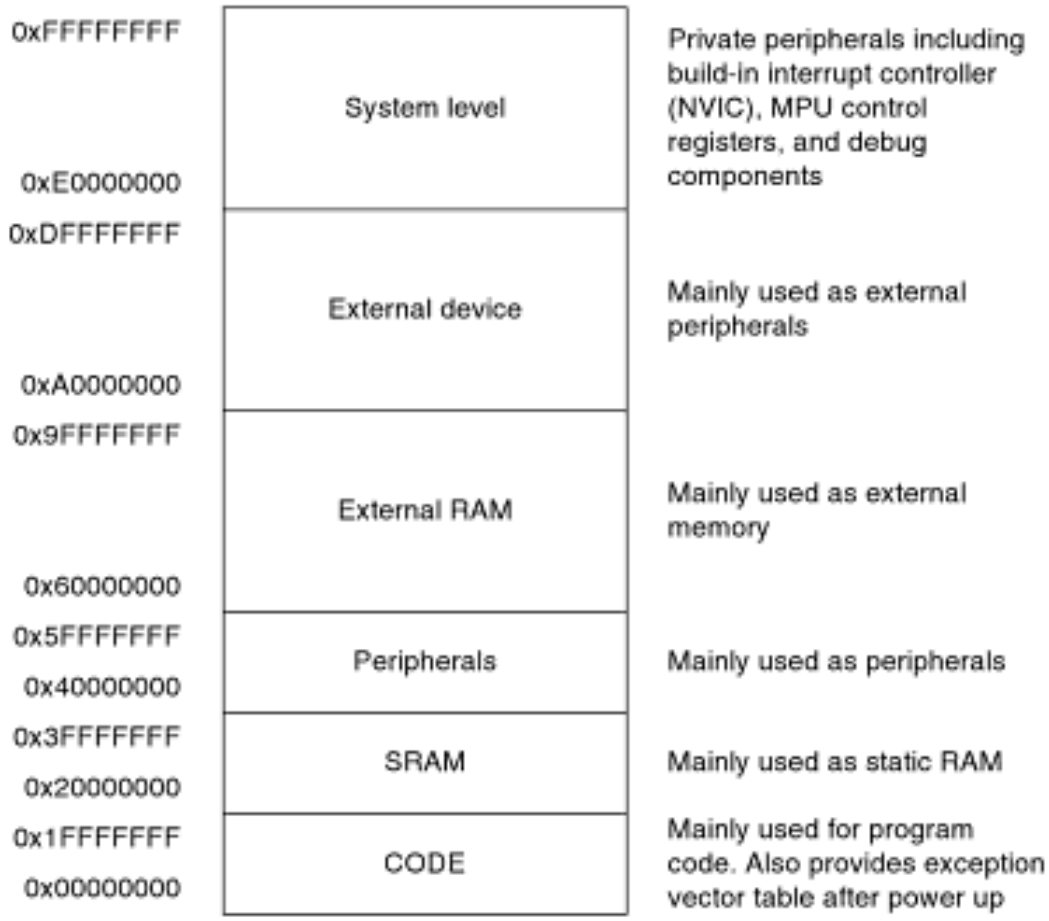
# ARM Organization

- In order to be processed data has to be moved from memory location to central set of registers
  - After processing data is stored back in memory
- Register bank connected to ALU via two datapath busses A and B
  - Bus B goes through Barrel shifter
    - It pre-processes data from source register by shl, shr or rotation
  - PC is part of register bank responsible for generating addresses for next operation
  - Registers can handle both data and address
- Address Incrementer block increments or decrements register values independent of ALU



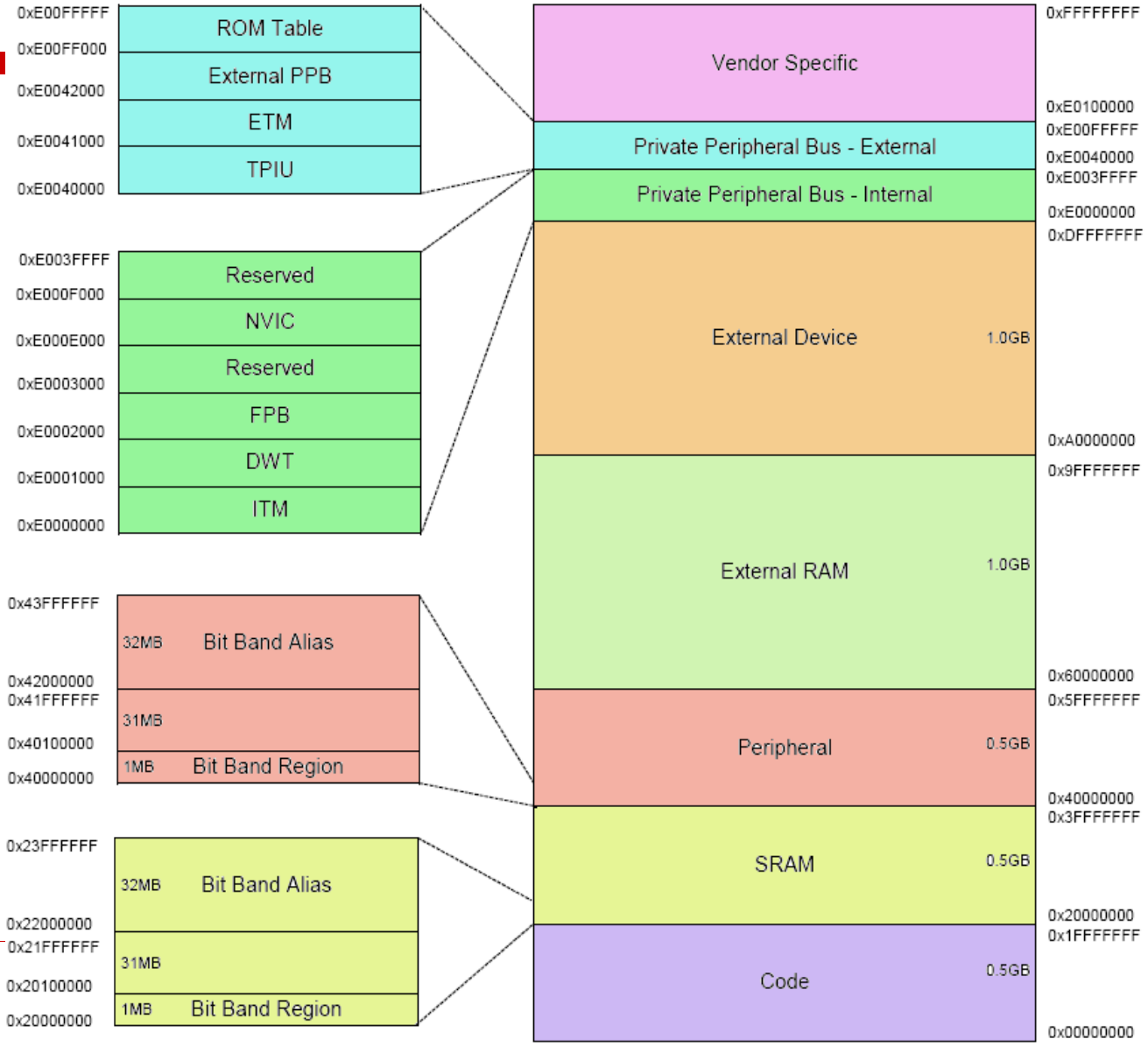
# ARM Cortex M Address Space

- 4 GB address space (ROM/Flash for Code)
- Built-in and external RAM
- Built-in and external IO
  - Memory-mapped IO
  - Special Function Registers (SFRs) accessed by processor



# Map -More

- Fixed memory map



# Memory Regions

- Program code located in code, SRAM or external RAM regions
  - Best to put program code in code region as then the instruction fetches and data accesses are carried out simultaneously on two separate bus interfaces
- SRAM memory (0x20000000 – 0x3FFFFFFF) is for connecting internal SRAM
  - Regions is executable, hence a program can be copied here and executed from this part of memory
  - Access through system interface bus
- On-chip peripherals (0x40000000 – 0x5FFFFFFF) is a 0.5 GB block intended for peripherals
  - Code cannot be executed from this region

# Memory Regions

- Two slots of 1 GB memory space are allocated for external RAM and external devices
  - External RAM regions (0x60000000 – 0x7FFFFFFF) and (0x80000000 – 0x9FFFFFFF) intended for either on-chip or off-chip memory. Code can be executed in this region
  - External devices (0xA0000000 – 0xBFFFFFFF) and (0xC0000000 – 0xDFFFFFFF) intended for external devices and/or shared memory. This is a non-executable region
- System region (0xE0000000 – 0xFFFFFFFF) – a 0.5 GB memory for system-level components, internal peripheral buses, external peripheral bus and vendor-specific system peripherals
  - It is a non-executable region